On domain decomposition with space filling curves for the parallel solution of the coupled Maxwell/Vlasov equations

Christian Konrad

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On domain decomposition with space filling curves for the parallel solution of the coupled Maxwell/Vlasov equations

Christian Konrad

N° 6693

October 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6693--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

On domain decomposition with space filling curves for the parallel solution of the coupled Maxwell/Vlasov equations

Christian Konrad *

Thème NUM — Systèmes numériques
Équipe-Projet NACHOS

Rapport de recherche n° 6693 — October 2008 — 97 pages

Abstract: Space filling Curves (SFCs) are increasingly used for combinatorial scientific computing and in particular for designing fast domain decomposition (partitioning) methods. In the context of parallel particle simulations for solving the system of Maxwell/Vlasov equations with a coupled FE/PIC (Finite Element/Particle-In-Cell) unstructured mesh based solver, one has to deal with a two-constraint partitioning problem. Moreover, this problem has to be solved several times during the simulation. Therefore, a fast and scalable partitioning problem is required. For this purpose, we propose here a new SFC based method which is well adapted to multi-constraint partitioning problems. This method is compared to graph based partitioning methods from the widely used MeTiS tool. Experimental results show that the proposed SFC based method is at least 100 times faster than MeTiS to the disadvantage of edge-cuts that are between 2 to 4 times worse than those achieved by the MeTiS methods.

Key-words: Domain decomposition, space filling curves, multi-constraint domain decomposition

* Stephané Lanteri

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Décomposition de domaine en utilisant des courbes de type Peano pour la résolution parallèle du système d'équations de Maxwell/Vlasov

Résumé : Les courbes de type Peano sont de plus en plus exploitées pour le calcul scientifique combinatoire et en particulier pour la mise au point de méthodes rapides pour la décomposition (le partitionnement) de domaine. Dans le contexte de simulations particulaires parallèles pour la résolution numérique du système d'équations de Maxwell/Vlasov par un solveur élément fini/PIC en maillage non-structuré, on doit traiter d'un problème de partitionnement bi-contraint. De plus, ce problème doit être résolu plusieurs fois au cours de la simulation, nécessitant une méthode de partitionnement rapide et scalable. Dans ce but, on propose ici une nouvelle méthode de partitionnement exploitant les principes des courbes de type Peano, qui est bien adaptée aux problmes de partitionnement multi-contraints. Cette méthode est comparée des méthodes de partitionnement de graphes mises en oeuvre dans l'outil MeTiS. Les résultats d'expériences numériques montrent que notre méthode est au moins 100 fois plus rapide que celles de MeTiS mais au détriment d'une dégradation de la taille des séparateurs de partitions d'un facteur 2 à 4.

Mots-clés : Décomposition de domaine, courbes de type Peano, décomposition de domaine multi-contraintes

1 Introduction

1.1 Combinatorial Scientific Computing

Combinatorial Scientific Computing (CSC) [13], a term coined in the last decade, denotes a research field that comprises combinatorial problems arising in computational science. In a typical setting, applied mathematicians develop numerical schemes for the solution of a Partial Differential Equation (PDE) and face computational questions during parallel implementations. Firstly, an appropriate *mesh generation* is required. A *domain decomposition* is needed in order to distribute mesh data among computational units. Further, for instance, in Finite Element Method (FEM) simulations, the *solution of a large sparse linear system* is necessary, often preceded by a *sparse matrix reordering*.

These problems exhibit all combinatorial features and may be the most prominent representatives of CSC. Bruce Hendrickson and Alex Pothen, researchers in CSC, provide an overview about CSC in their paper titled: "Combinatorial Scientific Computing: The Enabling Power of Discrete Algorithms in Computational Science" [13]. They emphasize the importance of *good* solutions and their impact on execution time of numerical applications, however problems in CSC seem also to have in common that the question of what is to be considered a *good* solution is highly problem dependant and it is hard or even impossible to answer this question once and for all. Even though it seems that for many problems we have a precise image in mind about the properties a good method should exhibit [1], for certain applications this way of assessing methods in question is insufficient. We discuss some problems that come with the assessment of methods for solving problems of CSC.

Firstly, most of the methods provide a compromise between quality of a solution and required computational resources (speed, memory). Considering again the problem of solving linear equation systems, some iterative methods for certain types of equation systems may result in poor accuracy to the benefit of being fast [2] while for instance direct methods may produce accurate results at the expense of high runtime.

Secondly, typically applications do not require methods that solve *any* possible problem instance but only the subset of problem instances that actually arise in the applications. Considering again the solution of linear systems in Finite Element (FE) simulations, for the fact that the arising systems are symmetric, a method does not need to be able to handle unsymmetric systems. A further example that is also the topic of the present report, is the problem of Domain Decomposition (DD). Graph partitioning is often applied as a method for decomposing meshes. For this, meshes are transformed into their dual graphs which are subsequently decomposed. In our setting we consider tetrahedral meshes and the dual graphs of tetrahedral meshes consist of nodes of maximal degree 4. Hence if choosing to solve the DD problem by reducing it to a graph decomposition problem it is sufficient if

[1] For instance a method for solving linear systems should be accurate and fast, or mesh generation should produce *nicely shaped* elements.

[2] E.g. few iterations of Jacobi or Gauss-Seidel iterations.

the graph partitioning method is able to handle the subset of graphs that contain nodes of maximum degree 4.

Thirdly, it is not always evident how to measure the quality of a solution. This is especially true for DD problems and in particular for the DD problems that are dealt with in the present report. For simple DD problems there are at least two aims, that is each partition should consist of the same number of elements and the partitions should be shaped such that the surfaces of the partitions are minimal, which, in the case of data dependency among adjacent elements that are separated by the decomposition, minimizes the communication time among computational units. Methods for solving the problem provide a compromise between these needs, perfect balance mostly leads to surface sizes that are worse than solutions with a slight imbalance. A good solution hence should consider the underlying hardware architecture of the simulation. If the interconnection network is slow, high emphasis should be put on the surfaces sizes of partitions whereas the communication is fast, good balancing is more important than small surface sizes.

In general, we search for solutions according to models we create. We can create continuously more complicated models that approximate the given situation. Pursuing the DD problem as a first step, we could investigate the correspondence of surface sizes to communication time. The so-called *edge-cut*, which represents the number of edges that are cut in the dual graph of a given mesh by a decomposition, is at present the dominating quality measure for DDs. Hendrickson and Kolda discuss in [12] that minimizing the edge-cut is the wrong measure if we aim to minimize communication time and they suggest the use of hypergraphs (which complicates our model to a high extent). Further, we could include heterogeneity in our model, for instance if we execute a numerical application on clusters with different types of processors. Even communication times among different computational units may differ. Communication among processor P_i to P_j might be fast while communication among P_i to P_k might be slow (due to the underlying interconnection network). In this setting high edge-cuts among the partitions for processor P_i and P_j are not costly while high edge-cuts among partitions for processor P_i and P_k are. These examples illustrate that it is a matter of discussion up to what extent it is reasonable to model a specific situation. The intention of these examples is to illustrate that in order to evaluate methods it is necessary to compare them according to certain measures (e.g. the edge-cut), however we always have to be aware up to what extent the measure models the real situation.

Fourthly, the fact that some of the problems of CSC are NP-complete makes theoretical considerations even more difficult. Even if we succeed in strictly defining a best solution to a given problem, for instance as a minimization problem, [3] and finding the best solution is NP-complete, we have to be aware that approximation algorithms that solve the problem are only heuristics and in many cases we are far from best solutions. Well-known problems are partitioning problems and graph coloring problems.

These aspects point out that attention has to be paid when discussing the applicability of methods for problems of CSC. This is especially true when we assess these methods theoretically. In many cases estimations can not be made at all and different methods have to be tested and compared.

[3] We may regard a best solution to the domain decomposition problem as the solution that minimizes a weighted sum of the imbalance of the partition sizes and the maximal surface size of a partition.

The present work considers the problem of DD for certain types of particle simulations. All of the four above mentioned criteria are met within this problem:

- Methods for solving the decomposition problems pose compromises between runtime and quality.
- We can restrict the set of decomposition problems to a certain subset of decomposition problems, that is certain types of so-called one-constraint and two-constraint decomposition problems as described in section 2.7.4.
- It is hard to establish models for characterizing the quality of solutions, compare this fact with our loose definition 2.10 of the DD problem.
- The decomposition problem is NP-complete.

The present technical report discusses DD for particle simulations for the solution of the coupled system of Maxwell/Vlasov equations and the design of a new DD method based on Space Filling Curve (SFC)s. In a first step, an analysis of the problem setting is done which leads to the formulation of the problem as so-called one-constraint and two-constraint DD problems. For our intended particle simulation we require solutions to two-constraint problems. These problems can be solved by multilevel graph partitioning. Since computation time is a crucial demand on the DD method, we have developed a new method based on SFCs.

1.2 Outline of the report

Chapter 2 introduces the coupled system of Maxwell/Vlasov equations (2.1), their discretization schemes (2.2, 2.3) and discusses the Particle-In-Cell (PIC) method that numerically couples the equations (2.4). We investigate on the demands of a DD for the PIC method (2.5) and provide a formal section for the definition of DD problems (2.6). The chapter is closed by section 2.7 that maps the requirements on a DD of the PIC method to the formal model introduced in section 2.6, in particular it points out that certain PIC simulations pose a two-constraint decomposition problem.

In Chapter 3 we discuss multilevel graph methods (3.1) as a method for solving the arising decomposition problems. We further discuss SFCs (3.2), that suit, due to their low running times, fine for so-called one-constraint DD problems. This chapter is also intended to provide the basic understanding of SFCs which is necessary for discussing our SFC based DD algorithm which is described in the subsequent chapter. We point out in section 3.2.5 that there is no straightforward extension of the applicability of SFCs to so-called two-constraint decomposition problems, which we aim to solve since they pose an appropriate model for certain PIC simulations.

Chapter 4 describes the main contribution of this work, the development and discussion of an SFC-based approach to solving two-constraint decomposition problems. The section is kept theoretical and technical details are postponed to the subsequent chapter.

In chapter 5 we discuss some technical aspects of implementation of our new method. This includes the computation of the inverse of SFCs (5.1) as well as the construction of what we call problem-specific SFCs (5.2).

In Chapter 6 we provide test cases. We compare the results obtained with our new method to results of methods provided by MeTiS [4], a state-of-the-art software package for DD following a multilevel graph-based approach, and outline that our method is extremely fast to a reasonable loss of quality.

We conclude with chapter 7 and discuss the sequel of this work.

[4] http://glaros.dtc.umn.edu/gkhome/views/metis

2 Problem formulation

This chapter introduces the Maxwell-Vlasov system and explains the PIC method as the method of discretization. The PIC method allows us to discretize and solve the Maxwell equations and the Vlasov equation independently, and consists of two *coupling phases* that connect these equations. For this reason the discretization issues of the two equations are presented independently. The numerical details of the Vlasov equation are presented because they contribute to the understanding of the particle-push phase of the PIC simulation. The discontinuous Galerkin method as the discretization scheme of the Maxwell equations is discussed as detailed as it is necessary for the arising DD problems. We continue with presenting the PIC method. A section about the difficulties of implementing the PIC method in parallel with a distributed memory system outlines that the parallel performance is mainly determined by an appropriate data distribution. The chapter is closed with definitions of the DD problems that are dealt with in the present report and a section that discusses how they are applied to the concrete setting of a PIC simulation.

2.1 The Maxwell-Vlasov system

An introduction to electromagnetic field theory is given by Thiedé in [32]. Details about the Vlasov equations are provided for instance in [3].

The Maxwell equations are a system of PDEs which describe the temporal evolution of an electromagnetic field induced by currents and charges:

$$\begin{aligned} \nabla \cdot (\overline{\overline{\epsilon}} E) &= \rho \quad \text{(Gauss's Law)}, && (2.1)\\ -\nabla \times E &= -\overline{\overline{\mu}} \frac{\partial H}{\partial t} \quad \text{(Maxwell-Faraday equation)}, && (2.2)\\ \nabla \cdot (\overline{\overline{\mu}} H) &= 0 \quad \text{(Gauss's Law for magnetism)}, && (2.3)\\ \nabla \times H &= j + \overline{\overline{\epsilon}} \frac{\partial E}{\partial t} \quad \text{(Ampère's Circuital Law)} && (2.4) \end{aligned}$$

with electric field E, the magnetic field H, the charge density j, the current density ρ, and the spatially varying tensors $\overline{\overline{\epsilon}}$ and $\overline{\overline{\mu}}$ the permittivity and the permeability. These equations form a hyperbolic system and given initial and boundary conditions they completely determine the temporal evolution of E and H.
We assume that the initial conditions E_0 and H_0 fulfill the properties:

$$\begin{aligned} \nabla \cdot (\overline{\overline{\epsilon}} E_0) &= \rho \quad \text{and} && (2.5)\\ \nabla \cdot (\overline{\overline{\mu}} H_0) &= 0, && (2.6) \end{aligned}$$

that means that the Gauss Laws 2.1 and 2.3 are initially fulfilled. If this is the case equations 2.1 and 2.3 are equivalent to the charge conservation law:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot j = 0. \qquad (2.7)$$

Hence, if we chose appropriate initial conditions and we ensure the conservation of charge, the four Maxwell equations reduce to the two equations 2.2 (Maxwell-Faraday equation) and 2.4 (Ampère's equation) which is a foreseen property for the applied discretization scheme.

These equations are non-linearly coupled to the Vlasov equation which states a condition for a charged-particle distribution $f(x, v, t)$:

$$\frac{\partial f}{\partial t} + v \cdot \nabla f + \frac{F}{m} \cdot \frac{\partial f}{\partial v} = 0, \tag{2.8}$$

with velocity v, spatial coordinate x, force F and particle mass m. The distribution $f(x, v, t)$ states the particle distribution in phase-space (x, v) and represents a usual continuous probability distribution.
The coupling is hidden in the force term F of the Vlasov equation. Since we model particles in an electromagnetic field the Lorentz force:

$$F = q(E + v \times B), \tag{2.9}$$

acts on each charged particle and the Vlasov equation transforms to:

$$\frac{\partial f}{\partial t} + v \cdot \nabla f + \frac{q}{m}(E + v \times B) \cdot \frac{\partial f}{\partial v} = 0. \tag{2.10}$$

For this coupling, we introduced electromagnetic quantities into the Vlasov equation. This can be thought of regarding the electromagnetic field as *given* and, as a consequence of the field, this induces forces acting on the particles. This cause-and-effect dependency can also be seen vice versa and we can also regard the electromagnetic field as a consequence of moving charged particles, that is we introduce the particle distribution f into the Maxwell equations. A snapshot of the particle distribution states the charge density and the current density which serve as input for the Maxwell solver. ρ and j are functions of x and t and are computed as the first and second moments of f:

$$\rho = q \int_{\mathbb{R}^3} f dv \tag{2.11}$$

$$j = q \int_{\mathbb{R}^3} v f dv. \tag{2.12}$$

2.2 Discretization scheme for the Vlasov equation

An introduction to particle methods is given in [3], in particular a discussion of the applied discretization method of the Vlasov equation can be found there. Issautier et al. solve in [7] the copled system of Maxwell-Vlasov equations in 2D on unstructured meshes with a finite volume method for the Maxwell equations. A compact description of the discretization of the Vlasov equation is provided there.

Discretizing the probability distribution $f(t, x, v)$ is done by a deterministic particle method. We introduce discrete *superparticles* $(P_k)_{k=1...N_S}$, N_S the number of superparticles, each representing a set of multiple particles. A superparticle P_k bears its own constant mass w_k and charge q_k which means that we assume that the constitution of a superparticle

does not change throughout the simulation. This allows us to approximate f by the sum over Dirac contributions of each superparticle:

$$f(t,x,v) = \sum_{k=1}^{N} w_k \delta(x - x_k(t))\delta(v - v_k(t)), \tag{2.13}$$

with the position x_k, the velocity v_k and the weight w_k of a superparticle. x_k and v_k are determined by the differential system:

$$\begin{cases} \dfrac{dx_k}{dt} = v_k = \dfrac{1}{m_k}\displaystyle\int F_k dt, \\ F_k = q_k(E(t,x_k(t)) + v_k \times B(t,x_k(t))). \end{cases} \tag{2.14}$$

The second equation of 2.14 states the Lorentz force F_k that is induced by the electromagnetic field which causes the particles to move. Equation one of 2.14 is basically Newton's equation of motion. Note that the formulation of this system is a direct consequence of the Vlasov equation 2.8 (compare [3]) , that means Newton's law of motion is already anchored in the Vlasov equation.

Discrete counterparts of equations 2.11 and 2.12 compute to:

$$\rho(t,x) = e\sum_{k=1}^{N} \omega_k \delta(x - x_k(t)) \quad \text{and} \tag{2.15}$$

$$j(t,x) = e\sum_{k=1}^{N} \omega_k v_k(t) \delta(x - x_k(t)). \tag{2.16}$$

When ρ and j are defined at the mesh vertices some interpolation scheme has to be applied, details and further pointers can be found for instance in [8].

2.3 Discretization scheme for the Maxwell equations

The Maxwell equations are discretized with a discontinuous Galerkin method. For the moment we assume that the charge conservation 2.7 is fulfilled at any time which supersedes the two Gauss Laws of the Maxwell equations as already mentioned in 2.1. Hence the set of four Maxwell equations reduces to the set of two equations:

$$\begin{cases} -\nabla \times E = -\overline{\overline{\mu}}\dfrac{\partial H}{\partial t} \\ \nabla \times H = j + \overline{\overline{\epsilon}}\dfrac{\partial E}{\partial t} \end{cases}. \tag{2.17}$$

This set of equations is solved on an unstructured tetrahedral grid $T = \{\tau_i | i \in I\}$ with the tetrahedra $(\tau_i)_{i\in I}$, N the number of tetrahedra and I an index set, that labels the tetrahedra, such that the computational domain $\Omega = \bigcup_{i\in I} \tau_i \subset [0,1]^3$ [1].

[1] For the sake of simplicity we consider here the domain Ω to be a subset of the unit cube, since SFCs are initially defined on this domain. Note that this general setting can be established by dilation and translation.

A tetrahedron τ_i consists of 4 faces. For two tetrahedra $\tau_i, \tau_j \in T$ it holds either $\tau_i \cap \tau_j = \emptyset$ or $\tau_i \cap \tau_j \neq \emptyset$, in the latter case the intersection is a common face and we denote it by F_{ij}. We call the set $N_{\tau_i} = \{\tau_j \in T | \tau_i \cap \tau_j \neq \emptyset, i \neq j\}$ the neighborhood of τ_i.

E and H are approximated by $E_h, H_h \in V_h^k = \{f \text{ sufficiently smooth} \quad | \quad f|_{\tau_i} \in \mathbb{P}_k(\tau_i), i = 1 \dots N\}$. $\mathbb{P}_k(\tau_i)$ denotes the space of polynomial functions of maximum degree k defined on τ_i. In other words, we seek solutions in a space of functions that, restricted to a tetrahedron, are polynomials. This definition implies that we allow solutions to be discontinuous across tetrahedron boundaries.

E_h, H_h are sought as linear combinations of basis functions $\varphi_{ij} \in \mathbb{P}_k(\tau_i), i = 1 \dots N, j = 1 \dots d_i$. For each tetrahedron τ_i we fix a value d_i that indicates the local degrees of freedom and determines the number of test functions associated to tetrahedron τ_i.

We multiply 2.17 by test functions φ_i, integrate by parts over a tetrahedron τ_i, replace E, H by E_h, H_h and obtain the weak formulation:

$$\left\{ \begin{array}{rcl} \int_{\tau_i} \varphi_{ij} \cdot \overline{\overline{\mu}}_i \frac{\partial H_h}{\partial t} & = & \int_{\partial\tau_i} \varphi_{ij} \cdot (E_h \times n) - \int_{\tau_i} \nabla \times \varphi_{ij} \cdot E_h \\ \int_{\tau_i} \varphi_{ij} \cdot \overline{\overline{\epsilon}}_i \frac{\partial E_h}{\partial t} & = & - \int_{\partial\tau_i} \varphi_{ij} \cdot (H_h \times n) + \int_{\tau_i} \nabla \times \varphi_{ij} \cdot H_h - \int_{\tau_i} \varphi_{ij} \cdot j \end{array} \right. , \tag{2.18}$$

n denotes the unitary normal, pointing outward a tetrahedron.

The volume integrals in 2.18 can be evaluated *locally*, that is in terms of data dependency computing volume integrals $\int_{\tau_i} \dots$ requires only degrees of freedom connected to tetrahedron τ_i.

The boundary integrals in 2.18 perform the actual coupling between adjacent tetrahedra. We can rewrite $\int_{\partial\tau_i} \dots$ as $\sum_{\tau_j \in N_{\tau_i}} \int_{F_{ij}} \dots$ where the latter integrals state the dependencies between adjacent tetrahedra. More precisely, the computation scheme for evaluating E_h, H_h on a face F_{ij} states these dependencies. Since our solutions E_h, H_h are discontinuous through element faces we nevertheless have to assign unique function values to E_h, H_h on faces. We use totally centered fluxes, that is:

$$E_h|_{F_{ij}} \approx \frac{E_h|_{\tau_i} + E_h|_{\tau_j}}{2} \quad , \quad H_h|_{F_{ij}} \approx \frac{H_h|_{\tau_i} + H_h|_{\tau_j}}{2}. \tag{2.19}$$

This scheme combined with a leap-frog time integration scheme leads to an almost explicit computation scheme that only requires the inversion of small local mass matrices. Further details about the described method can be found in [25, 26].

2.4 The PIC Method

The PIC method [14, 3] is a method that basically simulates moving particles on a grid. The framework handles at the one side a list of superparticles as the discretization unit of the Vlasov equation as well as a computational grid for the solution of the Maxwell equations. It consists of a four-step computation cycle as shown in figure 2.1.

- *Scatter*:
  The set of superparticles $(P_k)_{k=1\dots N}$ with their locations $(x_k)_{k=1\dots N}$ and their velocities $(v_k)_{k=1\dots N}$ determine ρ and j by equations 2.15 and 2.16. In this manner each

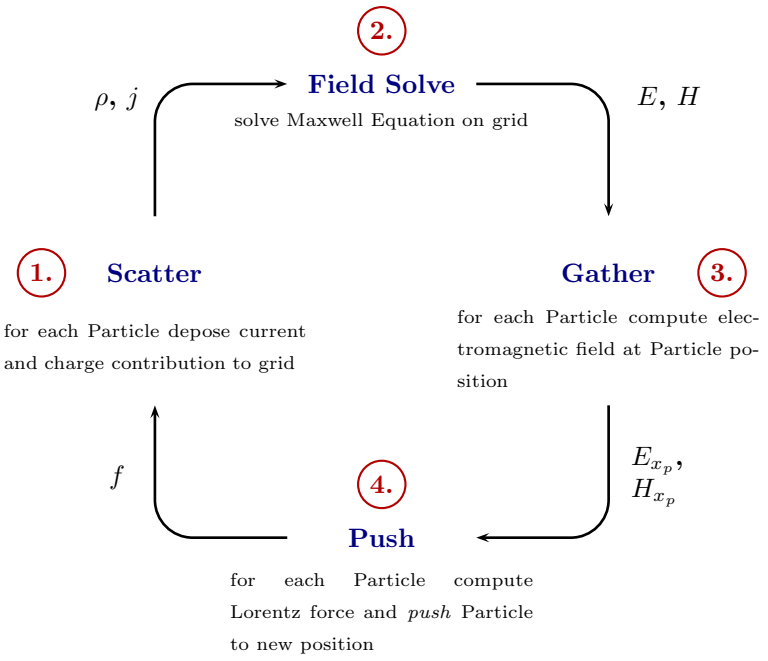


Figure 2.1: Main loop of a PIC simulation: the labels of the arrows denote the input/output quantities of the phases, E_{x_p} and H_{x_p} denote the electromagnetic field at a particle's location

particle contributes to the global current and charge density. Since ρ and j are discretized on the physical grid the contributions of each particle have to be *scattered* by some scattering scheme (compare e.g. [17]) to grid points in the neighborhood of the particle's location.

- *Field solve*:
  In the Scatter phase, ρ and j are set up and they serve as input for the field solve phase. The Maxwell equations are solved on the grid and produce the new electromagnetic field E, H.

- *Gather*:
  The electromagnetic field E, H is evaluated at the particle locations by an interpolation scheme. These values allow the computation of the Lorentz force acting on each particle.

- *Particle push*:
  By Newton's law of motion each particle is advanced due to the Lorentz force.

Note that this scheme contains two phases (particle push, field solve) that treat the two equations independently and two *coupling phases* (scatter, gather) that actually connect the two equations. The presented granularity of observation omits technical details. At this level the numerical schemes for the different phases are fully exchangeable. However, for a technical implementation, the interfaces between the phases can not be strictly separated; an

efficient particle simulation is a highly specialized tool, compare [6]. In an implementation the different phases might intertwine rather than being separately executed.

2.5 Parallelization issues

The parallel performance of a PIC simulation on a distributed memory system is mainly determined by the way data is distributed to the different processors, therefore we start with investigating on the two different kinds of data, the mesh data and the superparticles. We continue with pointing out the requirements of the different phases of the PIC simulation in terms of data distribution. We close this section with a distinction between independently implemented computational phases and interleavingly implemented phases.

2.5.1 Types of data

There are mainly two types of data that are manipulated throughout the particle simulation. Quantities like the electromagnetic field E, B, and the charge and current distribution ρ and j are mesh data and exhibit spatial order. On the other side there are the moving particles that can not be attached to any geometric position once for the entire simulation because they move throughout the simulation.

Data is distributed with respect to DDs. A *domain decomposition for the grid* states a mapping that assigns to each tetrahedron of the triangulation a processor number. The mesh data that is connected to this tetrahedron will be stored on the processor that is assigned to this tetrahedron. Since mesh data is connected to vertices and due to the fact that adjacent tetrahedra have a common face and hence common vertices, at partition boundaries a special treatment of mesh data is required. We do not consider this technical detail.

A *domain decomposition for the particles* also states a DD of the mesh and indicates the way we distribute the particles. All particles that reside in a tetrahedron that is assigned by this DD to a certain processor will be stored on the same processor as the surrounding tetrahedron. Note that, since the particles move, a particle might leave the cells that are stored on the same processor as the moving particle and we have to either *migrate* the particle to another processor (*particle migration*) or we implicitly adapt the DD for the particles by keeping the particle on the same processor.

Carmona and Chandler provide in [6] an overview about the structure of parallel PIC approaches. They point out that for a wide range of particle simulations the structure of data distribution can be described by the notion of a *particle box* and a *mesh box*. Each processor holds one particle box and one mesh box which correspond to the two partitions as a result of the DDs described above. Carmona and Chandler identify two different settings: in the first case a processor's particle box and its mesh box are different, and in the second case, they are not. For both cases they discuss the pros and cons. We decided to restrict our considerations to particle simulations with one single DD for both, the particles and the mesh, which corresponds to the case particle box = mesh box. This design decision will be justified after a discussion about the demands that the different computational phases ask for a good DD.

2.5.2 Requirements of the phases of the PIC method

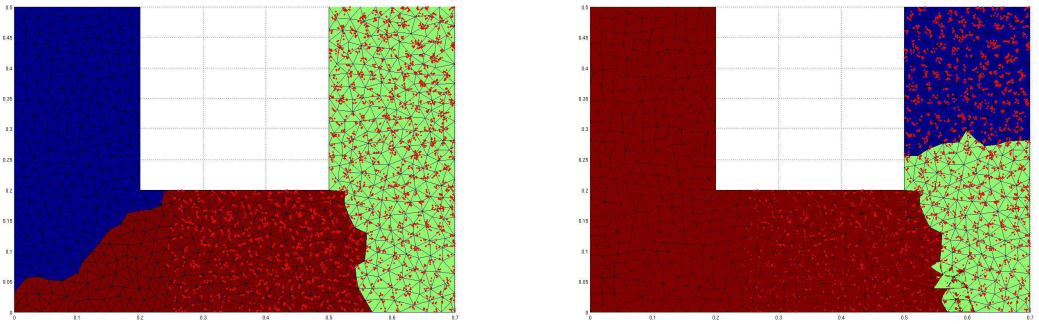


Figure 2.2: Left: good mesh box, Right: good particle box

Figure 2.2 shows a two dimensional decomposition problem of a U-shaped domain in 3 partitions. The density of the particles grows with increasing x and increasing y coordinate, hence the highest density is on the top right part of the domain.

The left image shows a good decomposition for the field solve phase. In this example the size of the partitions correspond to the workload for each processor (note that this is not always the case, see 2.7.1) which are well balanced in this example. Since the numeric scheme requires communication of boundary data among the processors the decomposition should be chosen such that the boundaries are small.

On the right side a DD for the particles is shown. The only property the particle push phase requires is that each partition contains the same number of particles because this balances the workload of pushing the particles. The example is one good solution.

Since the particles interact in the scatter and gather phases of the simulation with grid data that is geometrically close to the *current* position of the particle, it is beneficial in terms of data communication to place a particle onto that processor that also governs the cell that surrounds the particle. During the scatter phase the charge and current of a particle is deposed on the grid. We assume that the scattering scheme is local, that means that the particle's charge and current only contributes to mesh data associated to the tetrahedron that governs the particle. The same assumption holds for the gather phase. Note that this is not necessarily the case, see [17]. We assume that the electromagnetic field at a particle's position is interpolated from grid data connected to the sourrounding tetrahedron of a particle. These assumptions guarantee that there is no communication necessary during the gather and scatter phases.

Concerning the distribution of figure 2.2 this means that the illustrated situation of mesh box and particle box is only good for the green partition, all particles of the green partition (right image) reside in the mesh box of the green domain (left image), in other words, the green particle box is a subset of the green mesh box. This is neither true for the blue nor the red partition. It is the worst situation for the coupling phases of the blue partition since for all particles the blue partition has to initiate data communication with the green partition since the blue particle box is a subset of the green mesh box. To summarize,

for the coupling phases, an optimal situation is the setting that a processor's particle box matches its mesh box.

To avoid the communication overhead we decided to restrict us to the setting particle box = mesh box. In this case the scatter and gather operations do not require additional communication since it is ensured that the mesh elements reside on the same processor as the particle and we assume that both operations are *local*. From a computational point of view this also allows us to coarsen the 4-phases cycle of the simulation as a cycle with mainly two computational phases, the Maxwell solver (field solve) and a compound phase consisting of the particle-push, scatter and gather, since there is no communication required for the coupling phase and there is no processor synchronization needed during the compound phase. We will also refer to this phase as *extended particle push* phase.

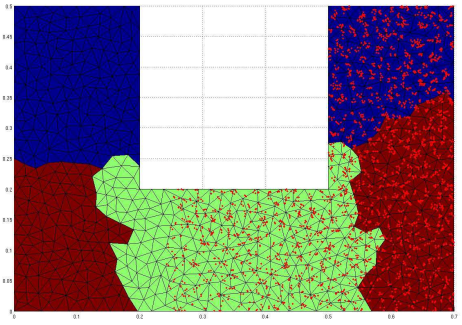


Figure 2.3: Good decomposition for both, the field solve and the extended particle push phase

The setting particle box = mesh box requires particle migration since we demand that a particle always resides on the processor that governs its sourrounding cells. Therefore even if we assume to have an initial decomposition that balances both (as in figure 2.3) the number of particles per processor and the sizes of the partitions, load imbalance will occur because the numbers of particles in the particle boxes change throughout the simulation. If we do not migrate the particles then we would implicitly adjust the particle box, hence this case corresponds to the setting particle box $\neq$ mesh box. In the latter setting the number of particles in a particle box would stay constant but the moving particles require communication during the coupling phases.

To cope with the arising imbalance by particle migration we consider Dynamic Domain Decomposition (DDD), that is we measure the load imbalance during the simulation and initiate a redecomposition if necessary. [34] considers load imbalance measures for particle simulations and decision criteria for the best moment to redecompose. Note that this demands from the decomposition strategy not only to produce good decompositions but also to be fast since we redecompose *online* (during the simulation).

We summarize these considerations in table 2.1 and the following statement:

Assuming an initially balanced setting, the fact that particles move in the setting of

computational phase	**requirement**
field solve	equally sized mesh boxes with small surfaces
particle push	equally sized particle boxes
gather + scatter	particle boxes match mesh boxes

Table 2.1: Desired properties of a DD for the different PIC phases

- particle box = mesh box implies particle migration. The boxes remain unchanged but the number of particles in the particle boxes change and this induces imbalance for the particle push phase.

- particle box $\neq$ mesh box leads to an implicit adaption of the particle box. The numbers of particles in the particle boxes do not change. No imbalance is induced for the particle push phase but an increase of communication in the gather and scatter phase occurs.

2.5.3 Execution model of a PIC simulation

We have to consider whether the two remaining computational phases, the Maxwell solver and the extended particle push phase are implemented as *independent* or *interleaving* computational phases, see figure 2.4.

In the independent setting the processes are synchronized after each computational phase while in the interleaved setting a processor already starts with the extended particle push phase as soon as he finished the solution of the Maxwell equations in its domain. The setting of independent computational phases arises for instance if an existing parallel Maxwell solver is used in the context of a PIC simulation. Implementing the interleaving setting requires intensive use of asynchronous communication of boundary data during the field solve and is from a practical point of view more difficult to implement than the independent setting. But note that an interleaving setting reduces not only idle time as shown in figure 2.4 but also poses as described in 2.7.4 an easier DD problem. In the interleaved setting we require to balance only one weight, that is the sum of the workload of the field solve phase and the workload of the extended particle push phase. In the setting of two independent computational phases a DD requires to balance both, the workload for the field solve and separately the workload for the extended particle push phase.

2.6 Formal definition of the domain decomposition problem

This section introduces further notations to which we refer throughout the present report and formulates the DD problems.

Definition 2.1 *A **domain decomposition** of the mesh T in N_P parts is a mapping*

$$d : I \to \{1, \dots N_P\}. \tag{2.20}$$

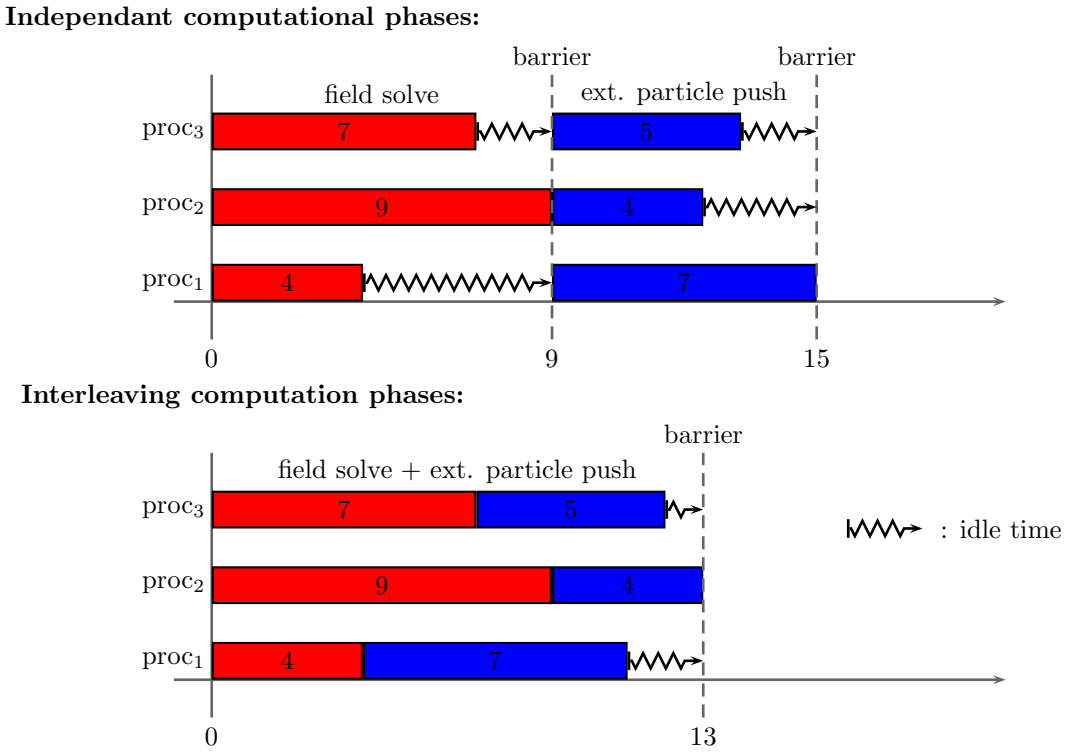


Figure 2.4: The computational phases of the PIC method can be implemented interleaved or independent

The sets $P_k = \{\tau_i \in T | d(i) = k\}, k \in \{1, \dots N_P\}$ *are the induced* ***partitions*** *by* d. $D(T, N_P)$ *denotes the set of all domain decompositions of the triangulation* T *in* N_P *parts. Note that* $|D(T, N_P)|$ *is finite* [2].

Assigning *weights* to the mesh allows us to model workload.

Definition 2.2 *A* ***weight*** ω *that is assigned to the mesh* T *is a mapping*

$$\omega : I \rightarrow \mathbb{R}_0^+, \text{ such that} \tag{2.21}$$

$$\sum_i \omega(i) = 1. \tag{2.22}$$

We write $\omega^i = \omega(i)$.

Note that assumption 2.22 is no restriction to the modelling. If a weight does not satisfy this property we simply divide each ω_i by $\sum_i \omega_i$. This is a common simplification also done for instance in [22].

In order to evaluate the quality of partitions we measure the balancing of the assigned weights, the edge-cut of the partitions as well as the edge-cut of the mesh.

[2] $|D(T, N_P)| = (N_P)^N$.

Definition 2.3 *The **edge-cut** of a partition P_k is defined as*

$$\mu_e(P_k) = \sum_{\tau \in P_k} |\{\tau_i \in N_\tau | \tau_i \notin P_k\}|. \tag{2.23}$$

Definition 2.4 *The **edge-cut** of a mesh T is defined as*

$$\mu_e(T) = \frac{1}{2} \sum_{P \in T} \mu_e(P). \tag{2.24}$$

Definition 2.5 *The **load** induced by a domain decomposition d for a partition P_k and the weight ω is defined as*

$$l(\omega, P_k) = \sum_{j \in P_k} \omega(j). \tag{2.25}$$

Definition 2.6 *The **balancing** of weight ω with respect to a domain decomposition d is defined as the quantity*

$$\mu_w(\omega, d) = N_p \cdot \max_i \{l(\omega, P_i)\}. \tag{2.26}$$

μ_w defines a total order on $D(T, N_P)$.

The definition of this measure follows the observation that we aim to minimize the *idle time* of a computational phase.

Definition 2.7 *Given a domain decomposition d and a weight ω then the **idle time** of the computational phase that is modeled by ω computes to:*

$$idletime(\omega, d) = \sum_{i=1 \ldots N_P} \max_j \{l(\omega, P_j)\} - l(\omega, P_i). \tag{2.27}$$

idletime defines a total order on $D(T, N_P)$.

The following theorem shows that idletime and μ_w order the set of domain decompositions in the same manner and hence justifies the definition of μ_w.

Theorem 2.8 $idletime(\omega, d_1) \leq idletime(\omega, d_2) \Leftrightarrow \mu_w(\omega, d_1) \leq \mu_w(\omega, d_2)$.

Proof: We consider $d_1, d_2 \in D(\omega, N_P)$. $(P_k)_{k=1 \ldots N_P}$ are the induced partitions by d_1, $(Q_k)_{k=1 \ldots N_P}$ are the induced partitions by d_2.

$$\begin{aligned}
\text{idletime}(\omega, d_1) &\leq \text{idletime}(\omega, d_2) \quad \overset{2,27}{\Leftrightarrow} \\
\sum_{i=1 \ldots N_P} \max_j \{l(\omega, P_j)\} - l(\omega, P_i) &\leq \sum_{i=1 \ldots N_P} \max_j \{l(\omega, Q_j)\} - l(\omega, Q_i) \quad \Leftrightarrow \\
N_P \cdot \max_j \{l(\omega, P_j)\} - \sum_{i=1 \ldots N_P} l(\omega, P_i) &\leq N_P \cdot \max_j \{l(\omega, Q_j)\} - \sum_{i=1 \ldots N_P} l(\omega, Q_i) \quad \overset{2,22}{\Leftrightarrow} \\
N_P \cdot \max_j \{l(\omega, P_j)\} &\leq N_P \cdot \max_j \{l(\omega, Q_j)\} \quad \overset{2,26}{\Leftrightarrow} \\
\mu_w(\omega, d_1) &\leq \mu_w(\omega, d_2).
\end{aligned}$$

■

idletime and μ_w define equivalence classes in $D(T, N_P)$. Elements in $D(T, N_P)$ are equal with respect to idletime or μ_w if they have the same maximum of the induced load for a partition. As an example considering again the field solve phase of the independent setting in figure 2.4, the splitting $7, 9, 4$ is *equivalent* with respect to idletime and μ_w to any other splitting with maximum 9, for instance $9, 2, 9$.

With these definition we state now the domain decomposition problems that are handled in the following chapters.

Problem 2.9 *Given a weight ω the **single-constraint domain decomposition problem** is the problem of finding a $d \in D(T, N_P)$ such that:*

$$max\{\mu_e(P_i)\} \quad is \quad small, \quad and \tag{2.28}$$
$$\mu_w(\omega, d) \quad is \quad small. \tag{2.29}$$

Problem 2.10 *Given multiple weights ω_i the **multi-constraint domain decomposition problem** is the problem of finding a $d \in D(T, N_P)$ such that:*

$$max\{\mu_e(P_i)\} \quad is \quad small, \quad and \tag{2.30}$$
$$\forall i \quad \mu_w(\omega_i, d) \quad is \quad small. \tag{2.31}$$

These DD problems are loosely formulated, especially note that we do not establish any relation in terms of importance about small edge-cut and good balancing(s), because, as section 2.7.5 points out, the requirements depend on the problem setting. Chapter 3 outlines that technical methods to solve problem 2.9 and problem 2.10 emphasize the conditions edge-cut and weight balancing differently. While multilevel graph partitioning methods aim primarily to minimize the edge-cut (objective) with the side conditions to balance the weights (constraints) this is vice versa with DDs along SFCs.

The following section describes how we apply this model to the settings of particle simulations.

2.7 Modelling particle simulations

The definitions in 2.6 give rise to defining two weights ω_M (**M**axwell) and ω_P (**P**articles) to model the computational load for the field solve phase and the particle push phase.

2.7.1 Modelling the load for the field solve phase

Concerning ω_M, due to the underlying numerical scheme (as described in 2.3, we have the possibility to vary the interpolation degree d_i at the element level) the computational load per element may vary. Clearly if we keep the same number of degrees of freedom for each tetrahedron we model $\omega_M = \frac{1}{N}$ since there is equal workload for each tetrahedron. If we vary the number of degrees of freedom per tetrahedron we have to figure out ratios of the computational loads for possible degrees of freedom that we apply. We do not precise here how to obtain good models but state brief hints. One way to obtain a first estimation to these ratios can be done theoretically by summing up, according to the numerical scheme,

for each occurring number of degrees of freedom the number of Floating Point Operation (FLOP)s, weighted by the number of tact cycles they need to perform, that are required to perform the computations for a tetrahedron. A practical and probably even more accurate alternative is plainly performing time measurings.

This modelling lets us assume that it holds $\omega_M > 0$ for all tetrahedra. We further assume that $\frac{\max\{\omega_M(i)\}}{\min\{\omega_M(i)\}} < C$, for some small C, that is we assume that the work load does not vary too intensively, for instance $C = 5$.

2.7.2 Modelling the load for the particle push phase

The computational load for the particle push phase is also modelled tetrahedron-wise. We assume that pushing a single particle requires computational load $\nu = \frac{1}{\text{total number of particles}}$ and a reasonable way to define ω_P is:

$$\omega_P(i) = \text{“Number of particles in tetrahedron i”} \cdot \nu.$$

Note that it is not realistic to restrict ω_P in any way, there might be tetrahedra that do not govern any particle at some time, hence $0 \in \omega_P(I)$. Further we can think of simulations where particles are attracted by a certain region in the computational domain and hence few tetrahedra will contain almost all particles. Therefore we do not impose any limitation on the number of particles per tetrahedron or any similar restriction as for ω_M.

2.7.3 Influence of the edge-cut for a particle simulation

A particle simulation benefits from a DD with a small edge-cut at least at two steps. As already discussed in 2.5.2 there is firstly communication of boundary data during the solution of the Maxwell equation. The edge-cut provides an estimation for the transfer volume hence it is beneficial to keep the edge-cut low. Secondly a small edge-cut is also beneficial for the particle push phase, more precisely the amount of particle migrations that are necessary during the simulation. Due to some Courant-Friedrichs-Levy condition (CFL)-condition for the solution of the Maxwell equations the time step in particle simulations is limited which leads to a limited distance a particle can travel during one particle push phase. Further, with decreasing edge-cut, the probability that the neighborhood of a particle's surrounding tetrahedron contains tetrahedra from other partitions decreases. For this reason the likelihood that a particle remains on a certain processor during one particle push phase increases with decreasing edge-cut. Less particle migration keeps a decomposition balanced and minimizes the communication required for distributing particles.

2.7.4 Execution models

As 2.5.3 points out we differ between interleaving computational phases and independent computational phases. The difference between the two models is reflected in the number of constraints we set up for the DD problem.

In the setting of independent computational phases we indeed model the problem with a two-constraint DD problem, balancing ω_M and ω_P.

In the interleaved setting we have to balance the sum of the loads for the two phases. That is here we model the problem with a single-constraint DD problem balancing the constraint $\frac{1}{2}(\omega_M + \omega_P)$.

2.7.5 Models for the runtime of parallel PIC simulations

Zumbusch provides in [36] a simple model for the runtime of a parallel FEM solver. We adapt and extend this model here for the interleaved setting and the independent setting.

The runtime t_{cyc} of one cycle of the PIC simulation is the runtime of the processor with the highest load. This runtime consists of computation time and communication time. The computation time is linear in the number of tetrahedra of a partition (which is linear to the global number of tetrahedra N) and linear in the number of particles of a partition (which is linear to the global number of particles $N_{particles}$). We assume that communication time is linear to the maximal edge-cut of a partition of the decomposition, which we assume is linear to the overall edge-cut of the decomposition. In the independent setting we obtain:

$$t_{cyl} = C_1 N N_{particles} \mu_w(\frac{1}{2}(\omega_M + \omega_P), d) + C_2 \mu_e(T), \tag{2.32}$$

for constants C_1, C_2. In the independent setting we obtain:

$$t_{cyl} = C_1 N \mu_w(\omega_M, d) + C_2 N_{particles} \mu_w(\omega_P, d) + C_3 \mu_e(T). \tag{2.33}$$

Further if we consider DDD the overall computation time t of the simulation grows linearly with the time t_d needed for redecompositions (and note that ω_P changes with the time).

$$t = C_1 N \mu_w(\omega_M, d) + C_2 N_{particles} \mu_w(\omega_P, d) + C_3 \mu_e(T) + C_4 t_d. \tag{2.34}$$

Equation 2.34 shows that the overall computation time depends on many aspects. We see that the importance of good balancings of ω_M and ω_P depends on the number of tetrahedra and the number of particles respectively. Since practical DD methods pose compromises between small edge-cuts and good balancings it is problem-dependant whether to optimize the balancing or the edge-cuts in order to achieve low computation times. Further we can also construct examples where the redecomposition time is more important than the balancing or the edge cut.

3 Domain decomposition methods

Multilevel graph partitioning algorithms are widespread methods for graph partitioning in scientific computing because they produce good quality partitions, they are robust and due to their hierarchic nature they are fast. To make them applicable, the *dual graph* (see 3.1.1) of a mesh is constructed and the decomposition is performed on the graph. There are several software packages (among MeTiS [1], Jostle [2], Chaco [3], Scotch [4]) that make them convenient to apply in scientific applications. These methods are also applied for particle simulations as for instance in [16]. The authors describe the application of PJostle, the parallel version of Jostle, in the scenario of a particle method with DDD on unstructured tetrahedral grids.

Not all graph partitioning software packages offer the possibility to balance multiple weights. MeTiS for instance supports true multi-constraint partitioning in the sense of our definition of the multi-constraint partitioning problem in 2.10, which is one reason why we decided to compare our results in chapter 6 to MeTiS. Jostle for instance comes with the so-called multi-phase partitioning which requires that the weights are 0 for at least some of the graph vertices. These methods are explained in 3.1.4.

Besides multilevel methods there are a variety of less known but though competitive approaches. For instance recently Meyerhenke *et al.* developed a diffusion-based graph partitioning heuristic based on shape optimization, whose edge-cuts are minimized by the objective to generate surface minimizing shapes. They describe their method in [27] and claim that their algorithm computes consistently better results than MeTiS and Jostle in terms of edge-cuts.

Mesh partitioning along SFCs are known to be very fast, which is an important property for DDD and the subject of the present work. In addition decompositions via SFCs result in a cache-efficient mesh data storage. Section 3.2 gives a brief introduction to SFCs and how they are applied for DD for single-constraint problems. The chapter is closed by section 3.2.5 that points out that SFCs are not directly applicable as a device for DD for 2-constraint problems as required for particle simulations with independent computational phases.

3.1 Multilevel graph methods

Before discussing the main principles of multilevel graph partitioning, we give a brief discussion about the construction of graphs from meshes.

[1] http://glaros.dtc.umn.edu/gkhome/views/metis/
[2] http://staffweb.cms.gre.ac.uk/~wc06/jostle/
[3] http://www.sandia.gov/~bahendr/chaco.html
[4] http://www.labri.fr/perso/pelegrin/scotch/

3.1.1 Constructing dual graphs

The vertex-weighted undirected dual graph $G = (V, E)$ to the triangulation T is constructed as follows. For each $\tau_i \in T$ we create a vertex, hence $V = T$. Further, for each τ_i and for each $\tau_j \in N_{\tau_i}$ we add the edge (τ_i, τ_j) to E if it is not yet contained. It is immediately clear that for unstructured tetrahedral meshes the degree of a vertex is limited by 4 (for unstructured triangular meshes the degree is limited by 3), the maximal number of faces an element can have. Figure 3.1 illustrates an unstructured triangulation of a triangle and its corresponding dual graph.

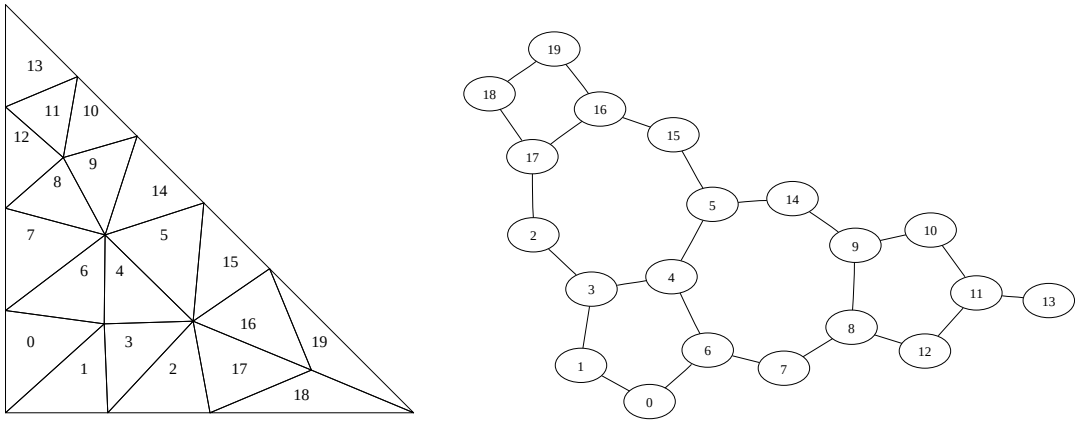


Figure 3.1: A triangular mesh and its corresponding dual graph. The degrees of the nodes of the dual graph of an unstructured triangular mesh is limited to 3.

This construction allows to transfer the defined weights directly to the graph, since weights are defined on elements and elements are transformed into vertices. Therefore we construct a vertex-weighted graph. Furthermore, a solution to the graph decomposition problem (a function that maps a vertex to a partition number) is directly interpretable as a solution to the mesh decomposition problem.

Note that a graph that is constructed from a mesh is a representation of the connectivity of the elements of the mesh and contains no geometrical information about the mesh. Graph-based mesh decomposition therefore just considers the connectivity of a mesh. Contrariwise, decompositions along SFCs do not consider the connectivity of a mesh but just the spatial location of elements.

3.1.2 Multilevel structure

Multilevel graph partitioning methods consist of three phases and follow the V-cycle as shown in figure 3.2.

In the *coarsening phase*, a series of successively coarser graphs is constructed. One coarsening step starts with identifying a matching, that is a set of edges, that do not have a common vertex, compare e.g. [28], of the fine level graph. The vertices that are connected

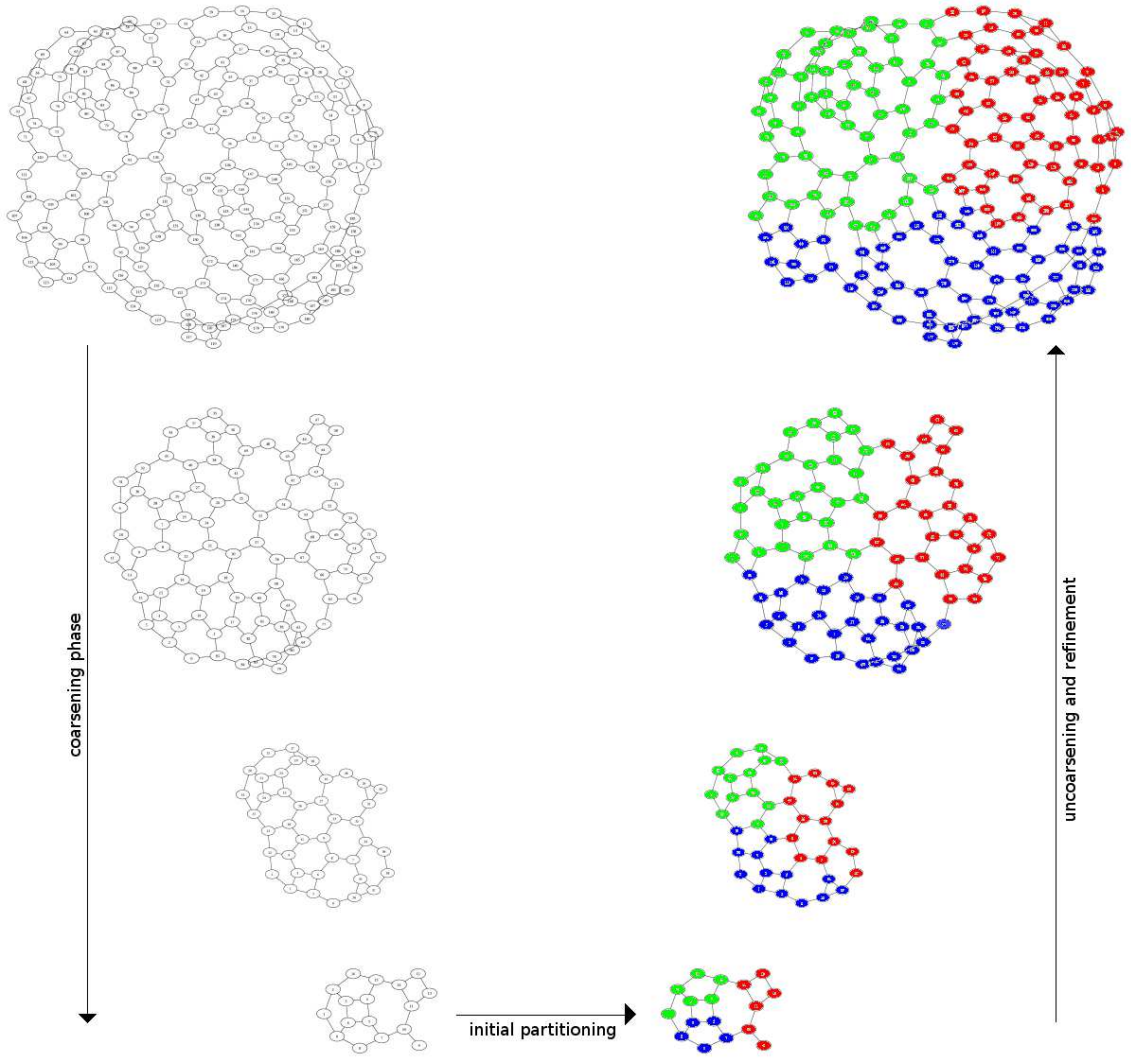


Figure 3.2: V-cycle of multilevel graph partitioning: multilevel graph partitioning algorithms coarsen the input graph, compute a solution on the coarse graph, and prolong and refine the coarse solution to obtain a solution for the initial graph.

by an edge of this independent set are collapsed and they form a single vertex in the coarse level graph and the weights of the vertices and edges are appropriately adapted.

In the *initial partitioning phase* the coarsest graph produced by this construction is decomposed with a conventional partitioning algorithm.

This initial decomposition is transfered to the finer levels in the *uncoarsening phase*. At each uncoarsening step a refinement heuristic is performed to improve the quality of the decomposition until the initial level is reached.

3.1.3 The single-constraint problem

As the graph partitioning problem is NP-complete, so are the objectives of the three phases of the multilevel approach. Finding matchings for arbitrary graphs is known to be NP-complete, the initial graph partitioning is NP-complete as well as the decomposition improvement step since the best decomposition improvement would be to solve the decomposition problem itself. For this reason it is necessary to apply heuristics for each of these three phases. This observation shows that a variety of algorithms can be constructed that follow the multilevel scheme. There are plenty of algorithms for finding matchings (Random Matching, Heavy-Edge Matching, Sorted Heavy-Edge Matching, ... compare [21]) as well as for improving existing decompositions (strategies based on the Kerninghan-Lin algorithm [23], boundary refinement strategies, ... compare [21]) and for performing an initial decomposition (Random initial decomposition, Region Growing approaches, ... compare [21]) each with characteristic weaknesses and potentials. For different partitioning problems different heuristics perform differently well, even a certain combination of heuristics for the three phases might perform well on certain inputs but produce bad results on others. For this reason MeTiS implements multiple strategies for these three phases and offers the possibility to the user to select the heuristics to apply in each phase.

For single-constraint problems graph partitioning software packages allow the user to specify an imbalance $\mu_w(\omega, d)$ which the algorithm tries to achieve and under which the algorithm tries to improve the edge-cut. Allowing a higher imbalance allows more possibilities for improving the edge-cut, hence multilevel graph partitioning poses a compromise between edge-cut and balancing.

3.1.4 Multiple constraints

To the author's best knowledge there are only two software packages, MeTiS and Jostle, that provide the possibility to balance multiple constraints. Karypis and Kumar, the developers of MeTiS, refer to this problem as *multi-constraint graph partitioning problem* in [22], that is where we adopted this term. The MeTiS team provides two formulations of the multi-constraint graph partitioning problem:

Problem 3.1 *The **horizontal multi-constraint partitioning problem** is the problem of, given real numbers $c_i \geq 1$, find a domain decomposition d such that the sum of the edges cut by the partitioning is minimized subject to the constraints:*

$$\forall i : \mu_w(\omega_i, d) \leq c_i, \tag{3.1}$$

and:

Problem 3.2 *The **vertical multi-constraint partitioning problem** is the problem of, given a real number $c \geq 1$ and real numbers $r_i \leq 1$ such that $\sum_i r_i = 1$, find a domain decomposition d such that the sum of the edges cut by the partitioning is minimized subject to the constraint:*

$$\sum_i r_i \mu_w(\omega_i, d) \leq c. \tag{3.2}$$

Problem 3.1 aims that each weight is balanced with respect to an imbalance limit chosen for each weight independently, while 3.2 aims to balance each weight such that a convex combination of the imbalances fulfills a certain limit. The horizontal formulation is more strict than the vertical formulation, however the vertical formulation allows more flexibility in the modelling of a problem. Consider for instance a particle simulation that spends 80% of its computation time in the particle push phase and only 20% in the field solve phase. For this problem a balancing of the weight that models the particle push (ω_P) impacts stronger on the total computation time than the balancing for the weight that models the field solve phase (ω_M). The total computation time for the simulation is proportional to $0.2 \cdot \mu_w(\omega_M, d) + 0.8 \cdot \mu_w(\omega_P, d)$ (according to this very simplified model). Hence, the setting $\mu_w(\omega_M, d) = 1.4, \mu_w(\omega_P, d) = 1.02$ computes in total faster than $\mu_w(\omega_M, d) = \mu_w(\omega_P, d) = 1.1$. Modelling this example with the vertical formulation of the multi-constraint problem yields domain decompositions that exhibit higher imbalances for some weights while the balancings of other weights recompense these surpluses.

For both formulations, the MeTiS group developed multilevel algorithms that follow the three phases scheme as presented above, that is the *multi-constraint multilevel recursive bisection* algorithm for the horizontal formulation of the problem and the *multi-constraint multilevel k-way partitioning* algorithm for the vertical formulation, both described in [22].

Walshaw and the Jostle group present in [33] their approach to balancing multiple weights which they call *multi-phase partitioning*. Their approach follows the horizontal formulation 3.1. For the fact that their algorithm requires that $\omega_i(\tau) = 0$ for at least some τ, their algorithm is only applicable for a subset of the problems that could be solved by MeTiS. To the author's best knowledge there are no papers comparing the multi-constraint decomposition abilities of MeTiS and Jostle.

3.2 Space filling curves

Before illustrating the capabilities of SFCs as a device for DD, we give a brief introduction to SFCs for a better understanding of the method. Chapter 5 describes the technical realization of our work and takes on this introduction.

The term "Space Filling Curve" arose with a graphical representation by David Hilbert in 1891 of a curve that fills the entire unit square. At this time mathematicians searched for a bijective mapping from the unit interval to the unit square which would have had a strong impact on the concept of dimensionality. Eugen Netto showed in 1889 that such a mapping does not exist, however dropping the requirement of injectivity, mathematicians formulated curves that are surjective but not injective. We define a Space Filling Curve as follows:

Definition 3.3 *A d-dimensional* ***Space Filling Curve*** *($2 \leq d < \infty$) is a continuous and surjective mapping*

$$f : [0,1] \rightarrow [0,1]^d . \tag{3.3}$$

We denote $[0,1]$ the index space and $[0,1]^d$ the geometric space.

3.2.1 Recursive construction principle

The SFCs we consider are constructable by a recursive scheme as it is illustrated for the two dimensional Hilbert curve f_h in figure 3.3. The Hilbert curve is the limit of the displayed procedure. For further information about the recursive nature of SFCs refer for instance to [2, 30].

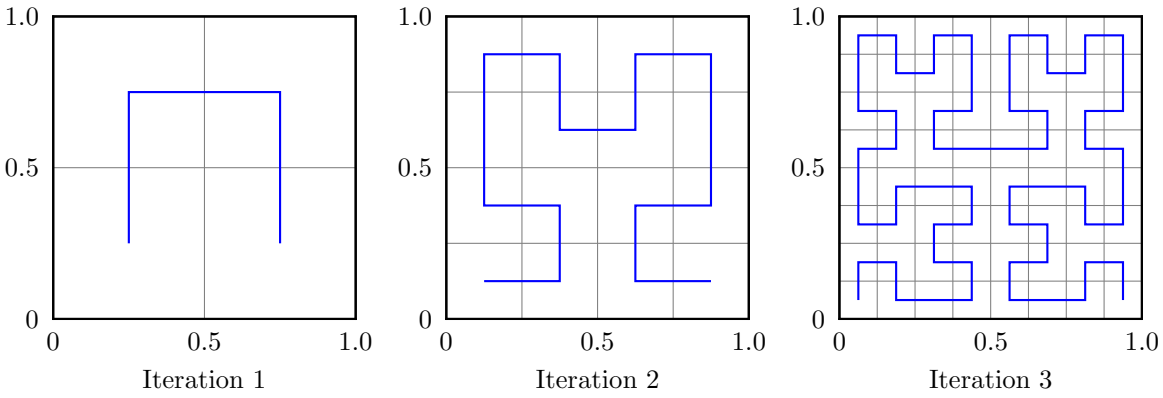


Figure 3.3: Recursive construction principle of the two dimensional Hilbert curve.

Iteration 1 consists of one *basic shape* of the curve which is used to construct the following iterations recursively. Figure 3.4 demonstrates this construction principle.

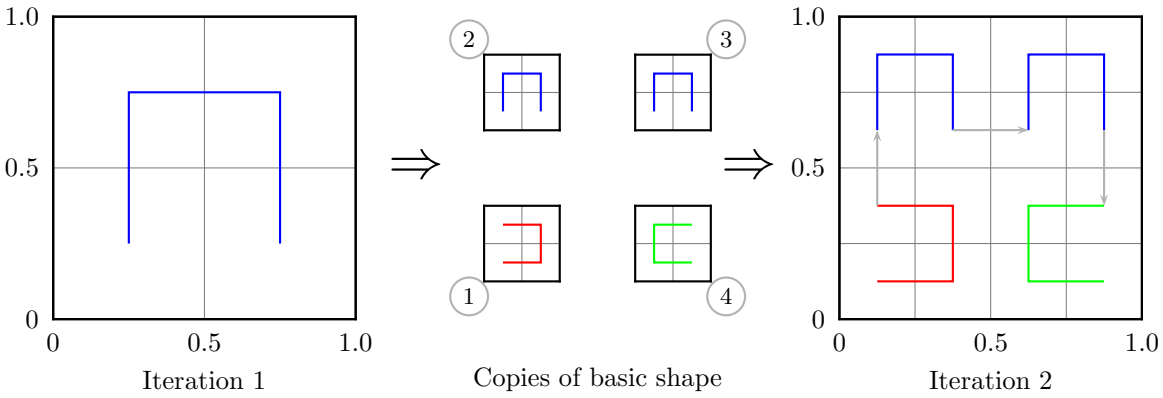


Figure 3.4: Constructing the second iteration of the Hilbert curve: the basic shape is quadruplicated, the parts are assembled and connected.

Besides ⊓ there are three further basic shapes ⊐, ⊏ and ⊔, which are rotated versions of ⊓. To construct a new iteration we substitute each basic shape contained in the prior iteration by four downscaled basic shapes and connect them appropriately.

In figure 3.4 the transition from iteration 1 to iteration 2 is shown. The blue basic shape (⊓) is substituted by the sequence ⊐, ↑, ⊓, →, ⊓, ↓, ⊏ in terms of its own direction [5], the arrows denote the direction of the connections between the basic shapes, hence iteration 2 can be seen as 4 basic shapes appropriately connected.

Constructing iteration 3 from iteration 2 follows the same scheme as the transition from iteration 1 to iteration 2 except that we have to substitute four basic shapes instead of only one. The first step to construct iteration 3 is to substitute the red basic shape (⊐) by four appropriately connected basic shapes. Considering figure 3.3 we construct a new substitution rule. We infer that ⊐ gets substituted by the sequence ⊓, →, ⊐, ↑, ⊐, ←, ⊔. In the same manner we construct a rule for ⊏ and with these two additional rules we can completely construct iteration 3.

Iteration 4 (which is not displayed here) defines a substitution rule for ⊔ in the same manner as we obtained the rules for the other three basic shapes. Note that these four iterations are sufficient to uniquely define the Hilbert curve in 2D. We summarize the obtained construction principle in table 3.1.

⊓	⇐	⊐	↑	⊓	→	⊓	↓	⊏
⊐	⇐	⊓	→	⊐	↑	⊐	←	⊔
⊏	⇐	⊔	←	⊏	↓	⊏	→	⊓
⊔	⇐	⊏	↓	⊔	←	⊔	↑	⊐

Table 3.1: Grammar for the 2D Hilbert Curve.

Using these rules as the productions of a context-free "L-system" (compare [29]), a context-free grammar with the constraint to apply as many rules as possible in each iteration, this system determines the iterations of the Hilbert curve and serves as a base for constructing *turtle graphics* (compare [2]), in principle a sequence of arrows that instructs a *turtle* into which direction it should move. Turtle graphics can be used to traverse two-dimensional data along the Hilbert curve as well as for simply constructing visual representations of the iterations of the Hilbert curve.

The following section points out that the recursive nature of the Hilbert curve gives rise to a computation principle of the mapping from index space to geometric space as well as its inverse.

3.2.2 Basic idea of an algorithm for computing the Hilbert mapping

The Hilbert curve in 2D exhibits an inherent quartering. Each basic shape is substituted by four basic shapes in each iteration. For actually computing the mapping from an index $i \in [0,1]$ to a 2D coordinate $c \in [0,1]^2$ we make use of this observation.

The first iteration of the Hilbert curve in figure 3.3 already shows the rough course of the curve: it starts in the left bottom subsquare (subsquare 0), then visits the left top

[5] We consider the curve's starting point in the bottom left subsquare, however this decision is arbitrary.

subsquare (subsquare 1) and the right top subsquare (subsquare 2) and ends in the right bottom subsquare (subsquare 3). Due to the local substitution principle and its recursive nature this gives rise to the assumption (and follows from a definition of the Hilbert curve by nesting intervals as e.g. in [2]) that once the curve exits a subsquare it never reenters again. Hence the curve visits all points in subsquare 0 before visiting any point of subsquare 1 (or even of any other subsquare).

Another important observation is that the Hilbert curve evolves with *constant speed* which formalizes as:

$$\int_{t_0}^{t_0+\delta} f_h(i) = \int_{t_1}^{t_1+\delta} f_h(i) \tag{3.4}$$

for $t_0, t_1, \delta \in [0,1]$ and $0 \leq a, b \leq 1-\delta$. That is if we interpret the index parameter of the Hilbert curve as a time parameter, then the curve travels in a certain time span always the same area independent from its absolute starting time.

These facts combined give a first estimation of the Hilbert mapping:

$$i \in \left[0, \frac{1}{4}\right] \quad \Rightarrow \quad c \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \tag{3.5}$$

$$i \in \left[\frac{1}{4}, \frac{2}{4}\right] \quad \Rightarrow \quad c \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \tag{3.6}$$

$$i \in \left[\frac{2}{4}, \frac{3}{4}\right] \quad \Rightarrow \quad c \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \tag{3.7}$$

$$i \in \left[\frac{3}{4}, 1\right] \quad \Rrightarrow \quad c \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right]. \tag{3.8}$$

Each further iteration allows us to restrict the interval for c. Let us assume $i \in \left[\frac{3}{4}, 1\right]$, hence c is in subsquare 3. As figure 3.3 shows the curve enters subsquare 3 on the top right subsquare of subsquare 3 and follows the green basic shape. We apply the same idea as for iteration 1 and we obtain the more precise estimation:

$$i \in \left[\frac{12}{16}, \frac{13}{16}\right] \quad \Rightarrow \quad c \in \left[\frac{3}{4}, 1\right] \times \left[\frac{1}{4}, \frac{2}{4}\right] \tag{3.9}$$

$$i \in \left[\frac{13}{16}, \frac{14}{16}\right] \quad \Rightarrow \quad c \in \left[\frac{2}{4}, \frac{3}{4}\right] \times \left[\frac{1}{4}, \frac{2}{4}\right] \tag{3.10}$$

$$i \in \left[\frac{14}{16}, \frac{15}{16}\right] \quad \Rightarrow \quad c \in \left[\frac{2}{4}, \frac{3}{4}\right] \times \left[0, \frac{1}{4}\right] \tag{3.11}$$

$$i \in \left[\frac{15}{16}, 1\right] \quad \Rrightarrow \quad c \in \left[\frac{3}{4}, 1\right] \times \left[0, \frac{1}{4}\right]. \tag{3.12}$$

This procedure may be continued until we reach the desired precision.

Representing the index i in base four system simplifies the computation of the Hilbert mapping. We explicitly denote the base four system by brackets and the subscript 4 while otherwise assuming the decimal system unless explicitly indicated. It holds

$$i = (0.i_0 i_1 i_2 \dots)_4 = \sum_k i_k \cdot (\frac{1}{4})^k, i_k \in \{0,1,2,3\}. \tag{3.13}$$

We benefit from the quaternary representation of i because each digit i_k of i directly determines a subsquare in the refinement of the above procedure. We illustrate this by the

example $i = 0.8 = (0.\underbrace{3}_{i_0}\underbrace{0}_{i_1}\underbrace{3}_{i_2}\underbrace{1}_{i_3})_4$. $i_0 = 3$ means that $i \geq 0.75$ and hence we already know that $f_h(i)$ is in subsquare 3. The next digit i_1 determines the subsquare in subsquare 3. Since $i_1 = 0$ we know that $i \leq \frac{13}{16}$ (cmp. equation 3.9) and hence $f_h(i)$ lies in the top right subsquare of subsquare 3. If for each refinement level we label the subsquares consecutively in the order the Hilbert curve passes through them, then the quarternary representation of the index directly tells us the sequence of subsquares that contain $f_h(i)$.

In section 5.1 we take on the quarternary representation of the index and describe our method for computing SFC mappings and their inverses.

3.2.3 Proximity preservation

DD along SFCs requires that the SFC preserves proximity, that is given two indices $i_1, i_2 \in [0, 1]$ that are *close*, for instance $|i_1 - i_2| < \epsilon$, we also require that their corresponding points in geometric space $f_h(i_1), f_h(i_2)$ are *close*, for instance $||f(i_1) - f(i_2)||_2 < \pi(\epsilon, (i_1 - i_2))$ with π being some appropriate dependency on ϵ and the distance from i_1 and i_2. The appropriate dependency that holds for SFCs is formalized in the definition of Hölder continuity.

Definition 3.4 *A mapping $f : [0, 1] \to \mathbb{R}^n$ is called Hölder continuous on the interval $[0, 1]$ to the exponent k if there is a $C \in \mathbb{R}$ such that:*

$$||f(i_1) - f(i_2)||_2 \leq C \cdot |i_1 - i_2|^k \quad \forall i_1, i_2 \in [0, 1]\,. \tag{3.14}$$

In general it holds that a d-dimensional SFC that can be recursively constructed is Hölder continuous to the exponent $1/d$, compare e.g. [2].

Note that there is no inverse statement of Hölder continuity for SFCs, that is given two *close* points p_1, p_2 in geometric space there is no statement that guarantees that their indices $f^{-1}(p_1)$ and $f^{-1}(p_2)$ are also *close* in some sense. This becomes immediately clear by considering points around the center $(0.5, 0.5)$, for instance for some $\epsilon > 0$ the indices of the point $(0.5 - \epsilon, 0.5 - \epsilon)$ that lies in subsquare 0 and the point $(0.5 + \epsilon, 0.5 - \epsilon)$ that lies in subsquare 3 have a distance of at least 0.5 in index space, compare equation 3.5.

3.2.4 The single-constraint domain decomposition problem

The basic idea of DD along SFCs is to reduce a multidimensional DD problem to a one-dimensional splitting problem. This holds for both, single-constraint problems as well as for our developed methods for two-constraint problems.

We illustrate the basic method with a 2D example based on a geometry that is similar to the geometry of the introductory example in figures 2.2 and 2.3. The U-shaped geometry as displayed in figure 3.5 consists of 2716 triangles, hence $T = \{\tau_1, \ldots \tau_{2716}\}$. The red overlay is the third iteration of the Hilbert curve.

For simplicity we assume one constant weight $\omega = \frac{1}{2716}$, that is we aim to decompose the mesh into partitions that contain the same number of triangles. We chose $N_P = 3$.

Reducing the dimensionality of the problem is done by the inverse of the Hilbert mapping f_h^{-1}, see section 5.1.1 for details about its construction. Since we deal with triangles and f_h^{-1} is defined on points, we have to assign points to each triangle. We use the *center of gravity*

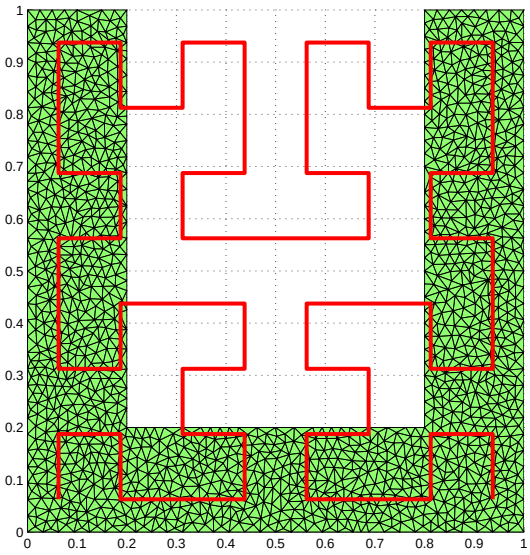


Figure 3.5: U-shaped example geometry with 2716 triangles, the red overlay is the third iteration of the Hilbert curve.

of a triangle, which has the property that it is, stochastically speaking, the expected value of the random experiment "pick some point from a triangle" and hence the best representative for the whole triangle:

Definition 3.5 *τ is a d-dimensional simplex with the vertices $x_1, \dots x_{d+1} \in \mathbb{R}^d$. We define:*

$$cg(\tau) = \frac{1}{d+1} \cdot (x_1 + \dots + x_{d+1}). \tag{3.15}$$

We compute $f^{-1}(cg(\tau)) \quad \forall \tau \in T$ and obtain 2716 indices as illustrated in the histogram of figure 3.6. In the following we may also omit the cg in the notation and abbreviate: $f^{-1}(\tau) := f^{-1}(cg(\tau))$.

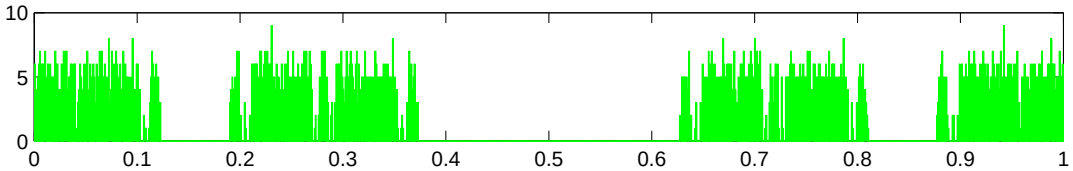


Figure 3.6: Hilbert index distribution of the triangles of example geometry 3.5.

The correspondence between the index distribution of figure 3.6 and the triangles of figure 3.5 becomes clear by regarding the overlain iteration of the Hilbert curve in figure 3.5. The curve starts in the bottom left corner, there are many triangles and hence in the interval from 0 to about 0.12 there are many indices. The gap from about 0.12 to about 0.18 corresponds to the exit of the Hilbert curve of the U-shaped geometry until its reentrance to

the left part of the U. In the same manner the other gaps can be explained. Note that since the Hilbert curve is axially symmetric to the straight line $y = 0.5$ and the triangulation of the geometry is nearly axially symmetric to the straight line $y = 0.5$ the index distribution exhibits approximately point symmetry to the point $x = 0.5$.

Now we perform the one dimensional splitting of the index distribution, that is we split the interval $[0, 1]$ into N_P parts and the triangles are distributed according to this interval splitting and their assigned Hilbert indices. Since we have a total weight of 1, each partition should have a weight of $\frac{1}{3}$ which corresponds to ~ 905 triangles ($\frac{1}{3}$ of the number of triangles). We sort the list of triangles increasingly with respect to their indices and the decomposition problem transforms to basically setting split points each 905 triangles. More generally, if we had an arbitrary ω, the splitting would be done by traversing the list of triangles and accumulating the weights of the triangles. A split point is set if the accumulated weight exceeds a multiple of $\frac{1}{3}$. This decomposition of the index space is visualized in figure 3.7.

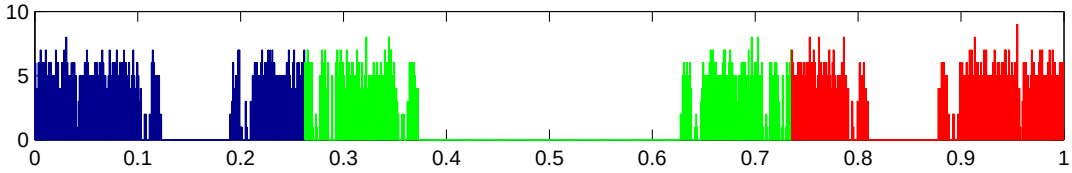


Figure 3.7: Decomposition result in index space.

The geometric counterpart as displayed in figure 3.8 illustrates that gaps in index space do not necessarily mean disconnected partitions. As figure 3.7 displays, the index space of each partition exhibits a gap but only the gap of the green partition leads to a disconnected partition.

It is immediately clear that this decomposition procedure enables to balance the weight *perfectly*, $\mu_w(\omega, d) = 1 + \epsilon$ for some small ϵ. That is due to the fact that at the procedure of setting split points it is very unlikely that the weights accumulate exactly to a multiple of $\frac{1}{N_P}$ and hence the load for a partition will always be a bit above or below $\frac{1}{N_P}$ which induces slight imbalance.

Note that this procedure does not allow any influence on the induced edge-cut. We split the unit interval into N_P parts such that each partition consists of one (connected) interval. And in this sense we can say that we minimize the edge-cut of the one dimensional problem. It is the SFC that transforms this decomposition to the multidimensional decomposition and the final quality of the decomposition depends on the curve. It is clear that the degree of *locality* of the curve affects the quality of the partition. There are few works discussing the term "locality" of SFCs, e.g. [9], and the quality of SFC induced partitions, e.g. [35].

3.2.5 Two constraints

There is no straightforward way of extending this procedure to balancing multiple weights that suits arbitrary decomposition problems. This section provides one *difficult* demonstrative problem instance for a two-constraint problem and discusses why a one dimensional

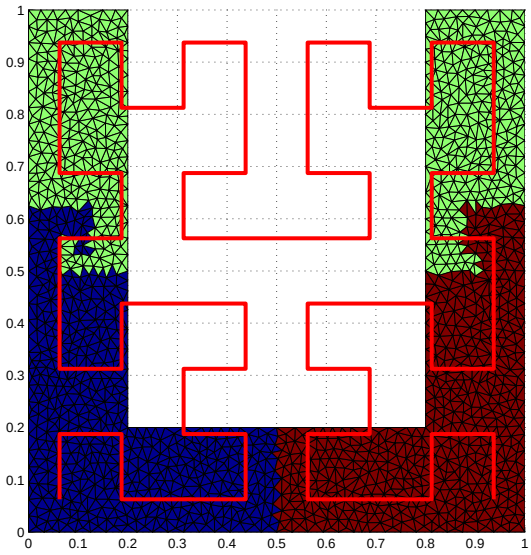


Figure 3.8: Decomposition result in geometric space.

interval splitting into N_P intervals is in general not sufficient for a balanced decomposition. Our method as described in chapter 4 is able to solve problems of this kind.

We extend the problem from the previous section, that is we consider the U-shaped geometry of figure 3.5 with one constant weight $\omega_M = 1/N$ which is displayed in figure 3.6 in index space. The M as the index of ω_M indicates that this weight models the computational load for the Maxwell-solver phase. We assume that this weight is constant for each triangle.

We add a second weight ω_P which represents a particle distribution. The support of ω_P is limited to the top right area of the geometry as displayed in figure 3.9.

We compute the particle distribution in index space. The upper weight distribution in figure 3.10 displays ω_P, the lower weight distribution displays ω_M and is hence a copy of figure 3.6.

This example demonstrates that there is no simple interval splitting that balances both ω_M and ω_P. A splitting that suits ω_P requires that each of the three partitions gets a fraction of the particles and hence the two inner boundaries lie within the index range of the particles (somewhere between 0.62 and 0.69). Therefore the first partition ranges from 0 to the first boundary ≥ 0.62 which means that partition one already gets more than half of the weight of ω_M.

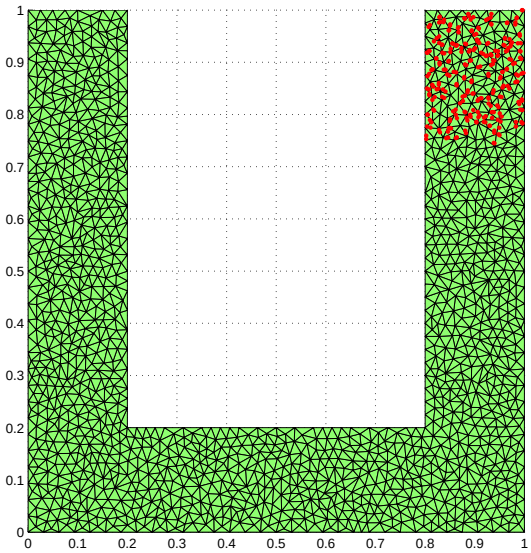


Figure 3.9: Two-constraint decomposition problem that can not be solved with SFCs by a simple interval splitting.

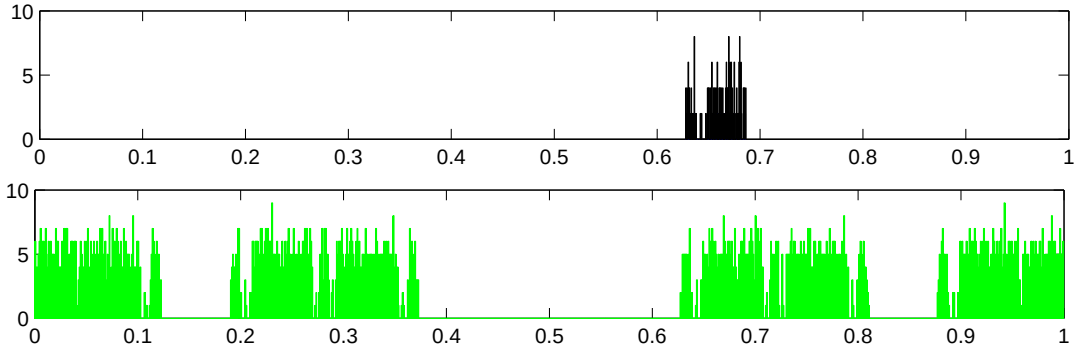


Figure 3.10: Two weights in index space.

3.3 Graph-based multilevel schemes versus space filling curves for one-constraint particle simulations

3.3.1 Weight balancing

As discussed in the prior sections decompositions along SFCs balance one single weight perfectly. The same holds for graph based methods, they achieve any user-defined imbalance. Hence both methods are equally good with respect to weight balancing.

3.3.2 Edge-cut

Schamberger and Wierum compare in [31] the edge-cut of mesh decompositions of MeTiS and SFCs induced partitions in the one-constraint setting. SFCs achieve edge-cuts between 1.2 up to 7 times worse than MeTiS. However, the cases where this factor exceeds 3 are rare. Furthermore, with the increase of the number of partitions this factor decreases.

This is the clear advantage of graph based methods.

3.3.3 Execution time and memory requirements

Typically, DD methods are part of a numerical application. It is not immediately clear what should be considered the execution time of a DD-method. If we consider the problem of, given a mesh, computing DD d, we also have to count the time graph based methods require for constructing the dual graph of a mesh. Furthermore, we have to consider the computation time of inverse SFCs and the time for sorting the mesh elements.

Now consider the interleaved setting of a particle simulation as discussed in section 2.5.3. We assume DDD (due to the imbalance induced by moving particles) and a mesh that does not change in time (no refinement for instance). If we apply graph based methods, then even though we redecompose during the simulation it is sufficient to construct the dual graph of the mesh once for the entire simulation, hence we should not consider this step.

Considering SFCs we can precompute the indices of the mesh elements of SFCs. Further we can already sort the mesh and store the mesh in a file already sorted with respect to their SFC indices. Hence during the simulation it is not necessary to compute any SFC indices or to perform sorting. Here a (re-)decomposition merely consists of setting new split points.

Papers, e.g. [31], underline that decompositions along SFCs are much faster than graph based methods even when the time for computing indices and the sorting are considered. Therefore it is expectable that for particle simulations with DDD a lot of time can be saved by the application of SFCs.

Concerning memory requirements, [31] states that graph based methods require much more memory resources than SFCs (they provide examples where MeTiS requires 220 MBytes of memory while SFCs require only 5MBytes, even though the computation of SFC indices and the reordering is considered).

3.3.4 Cache efficiency

Typically algorithms in scientific computing access mesh data not randomly. Discontinuous Galerkin methods, as applied in the discussed framework, compute numerical fluxes between adjacent tetrahedra, hence when accessing an element τ_i it is very likely that the elements

in N_{τ_i} are accessed sequentially. Therefore, a memory layout that stores the mesh in a proximity preserving manner is beneficial.

Proximity preservation is the attribute that enables SFCs to be used as a device for DD. Therefore we obtain cache efficiency for granted when decomposing along SFCs.

Unfortunately this is not true for decompositions via graph based methods. These methods require additional local reordering strategies (for instance based on SFCs) if cache efficiency is important.

3.3.5 Conclusion

As discussed in the introduction it is not possible to uniquely state a method that is superior. SFCs bring lots of advantages (execution time, cache efficiency) to graph based methods however they are inferior in terms of edge-cuts. These issues have to be equiponderated in concrete problem settings. Among others, the following questions have to be considered:

- Does cache efficiency matter?
- Is the interconnecting network fast/slow such that larger edge-cuts matter?
- Does memory usage matter?
- Do we redecompose?
- Will the mesh be refined?

4 Space filling curves for domain decomposition for two-constraint particle simulations

The two-constraint SFC DD is an interesting combinatorial problem. We introduce a more appropriate notation for this problem to highlight its combinatorial nature and define the two-constraint decomposition problem for SFCs. We show that an algorithm that splits the unit interval *optimally*, such that the weight balancing is satisfied *best possible*, in N_P parts, is in $O(N_P^{N_C} \cdot N^{2 \cdot N_C + 1} \cdot N_C)$, where N_C is the number of constraints, and hence for the two-constraint problem we obtain an algorithm that is in $O(N_P^2 \cdot N^5)$. This algorithm only suits for *simple* decomposition problems and is not generally applicable for the arising decomposition problems of a particle simulation. However, this algorithm poses some lower bound in terms of the number of cuts of the unit interval and can therefore be expected to produce good edge-cut results to the disadvantage of the weight balancing.

We continue by discussing our algorithm for the two-constraint decomposition problem and show that the algorithm can balance nearly any problem in charge of the edge-cut. Here we provide a worst-case quality bound.

We further discuss an arising subproblem of our algorithm, which we denote as the RE-UNIFICATION problem. This problem turns out to be NP complete and is structurally similar to the integer set partitioning problem. We provide a heuristic for REUNIFICATION that can be seen as an extension to the Karmarkar-Karp largest-differencing algorithm for the integer partitioning problem.

Our algorithm can be based on any SFC. However, depending on the SFC the difficulty of the problem varies. We develop new problem-specific SFCs that can *simplify* some decomposition problems. For these problems our two-constraint algorithm based on these problem-specific curves leads to smaller edge-cuts than the ones we obtain if our algorithm was based on the Hilbert curve.

4.1 Combinatorial problem

Throughout this chapter we assume N_C constraints which we denote by $(\omega_i)_{i=1\dots N_C}$. If we consider the two-constraint problem, we may also denote the two constraints with ω_M and ω_P. For each tetrahedron we compute their indices with the inverse of a SFC and sort the tetrahedra in ascending order with respect to their indices. We assume that the index set I for labeling the tetrahedra is exactly this ordering, that means for two tetrahedra τ_{i_1}, τ_{i_2} it holds: $i_1 < i_2 \Leftrightarrow f^{-1}(\tau_{i_1}) < f^{-1}(\tau_{i_2})$ for the applied SFC f. We also write $\omega^i = \omega(\tau_i)$.

We group the weights in the matrix W:

$$W \in (\mathbb{R}_0^+)^{N_C \times N} \tag{4.1}$$

$$W = \begin{pmatrix} \omega_1^1 & \omega_1^2 & \dots & \omega_1^N \\ \vdots & & & \vdots \\ \omega_{N_C}^1 & \omega_{N_C}^2 & \dots & \omega_{N_C}^N \end{pmatrix}. \tag{4.2}$$

As section 3.2.5 points out, most two-constraint decomposition problems can not be balanced by simply splitting the unit interval into N_P parts as in the single-constraint case. Therefore our strategy consists of two steps: we first split the unit interval into several ($> N_P$) intervals and in a second step we construct partitions by reuniting intervals.

In terms of the notation of W the first step of this procedure corresponds to finding vertical separators (here demonstrated with two constraints ω_M, ω_P)

$$W = \left(\begin{array}{ccc|c|c|cc} \omega_M^1 & \omega_M^2 & \dots & \dots & \dots & \dots & \omega_M^{N-1} \; \omega_M^N \\ \omega_P^1 & \omega_P^2 & \dots & \dots & \dots & \dots & \omega_P^{N-1} \; \omega_P^N \end{array} \right), \tag{4.3}$$

that group the discrete weights into several groups. We refer to these matrix subparts of adjacent weights in the following as *groups*. The groups are *reunited* afterwards into N_P partitions.

Before we present our considerations we give a remark on the definition of a weight in 2.6. We define a weight as a mapping from the set of tetrahedron indices to some real value. Sticking to this definition in the sequel is however quite cumbersome. For instance as part of a DD process we split the unit interval into subintervals and it will be necessary to compute the weight that falls to tetrahedra with an index in a certain interval $[a, b]$. A notation for this is:

$$\sum_{i \in \{j \in I | f_h^{-1}(cg(\tau_j)) \in [a,b]\}} \omega(i) \tag{4.4}$$

which is neither simple to read nor simple to write. There are multiple situations like these and for the sake of clarity we will extend the applicability of a weight to other mathematical objects as needed (and only if there are no ambiguities), for instance we will write for the upper expression simply $\omega([a, b])$. Since an interval characterizes in our setting a set of tetrahedra this extension is straightforward. The same holds for groups as defined in the prior subsection. Groups also denote tetrahedra and hence a group represents, in the same manner as an interval, a set of tetrahedra and we will also write loosely $\omega(G)$ for some group G.

4.2 Splitting W optimally into N_P parts

4.2.1 Motivation

Even though splitting the unit interval into N_P parts is insufficient for balancing the majority of two-constraint decomposition problems (e.g. the example in section 3.2.5), this idea is by all means theoretically interesting since we perform the least number of cuts possible. In this way a resulting partition consists of one single interval (in terms of the matrix W of

one group) and due to the Hölder continuity of the SFC it is very likely[1] that the partitions are connected in geometric space.

For this reason splitting W into N_P intervals provides an indication for the lower bound[2] for the edge-cut of a decomposition problem along the SFC. This implies that we can not expect better edge-cut results as in the case of a one-constraint problem. Increasing the number of splits and a subsequent reunion will hence increase the edge-cut of the resulting partition but offers the possibility to reunite the parts in a sophisticated way such that the balancing increases. In some sense the number of cuts of the unit interval is a parameter that trades off the edge-cut against the balancing of the weights.

4.2.2 Problem definition

A solution to the splitting of W into N_P parts is a vector $s \in I^{N_P-1}$ that indicates the indices after which we put a separator in W. We consider here the general problem with N_C constraints. We characterize the optimal solution s' as follows:

$$\mu_O(s) = \sum_{c=1}^{N_C} \max\{\sum_{i=0}^{s_1} \omega_c^i, \sum_{i=s_1+1}^{s_2} \omega_c^i, \ldots, \sum_{i=s_{N_P-1}}^{N} \omega_c^i\} \tag{4.5}$$

$$= m_{\omega_1} + m_{\omega_2} + \cdots + m_{\omega_{N_C}}, \quad \text{with} \tag{4.6}$$

$$m_{\omega_c} = \max\{\sum_{i=0}^{s_1} \omega_c^i, \sum_{i=s_1+1}^{s_2} \omega_c^i, \ldots, \sum_{i=s_{N_P-1}}^{N} \omega_c^i\}, \quad \text{then} \tag{4.7}$$

$$s' = \text{argmin}_s \mu_O(s). \tag{4.8}$$

$(m_{\omega_i})_{i=1\ldots N_C}$ denotes the maximal load of weight ω_i that is assigned to one partition.

We use the indices $(m_i)_{i=1\ldots N_C}$ to denote the partitions that pose the maxima $(m_{\omega_i})_{i=1\ldots N_C}$, for instance P_{m_4} is responsible for m_{ω_4}.

We say a solution $s \in I^{N_P-1}$ is well-formed if it represents a proper decomposition, for instance it guarantees that $s_i > s_{i-1}$ with $1 \leq i \leq N_P - 1$.

4.2.3 Proof: Problem is efficiently solvable

We start with a Lemma concerning the computational complexity of computing μ_O.

Lemma 4.1 *Given a well-formed solution $s \in I^{N_P-1}$ there is an algorithm that computes $\mu_O(s)$ in $O(N_C \cdot N)$.*

Proof: The matrix W contains $N_C \cdot N$ elements. Each element contributes exactly to one sum of μ_0 and is therefore only visited once. Selecting the maxima and summing up the maxima can be done on the fly while computing the sums.

[1] If the population of the intervals by triangles/tetrahedra is sparse or contains gaps, as for instance the green partition in figure 3.8, the partition may be disconnected. Note that even though the blue and the red partition of the example are also disconnected in index space their geometric counterpart is connected.

[2] At least it provides an indication for the order of magnitude of the lower bound.

■

Since there are $O(N^{N_P})$ different s, computing $\mu_0(s)$ for all s (which is simply an exhaustive search) leads to an algorithm with a complexity that grows exponentially with the number of partitions. To meet the above mentioned complexity of $O(N_P^{N_C} \cdot N^{2 \cdot N_C + 1} \cdot N_C)$, which is, for a fixed number of constraints, a polynomial complexity, we reconsider the definition of μ_0. μ_0 is a sum over the maxima of the loads for each constraint, hence a sum of N_C maxima. Each maximum belongs to one partition, and since there are N_P partitions there are $N_P^{N_C}$ different possible memberships of these maxima to the partitions. We will consecutively test all possible memberships of these maxima.

A partition is uniquely defined by two partition separators. In terms of the splitting of the unit interval it is defined by a lower bound and an upper bound. Therefore, given the membership of the assumed maxima, there are at most[3] $2 \cdot N_C$ partition separators to be set up in order to uniquely define the partitions that exhibit a maximum, therefore there are $O(N^{2 \cdot N_C})$ possibilities for setting the boundaries of the assumed maxima. Note that of course not all settings of separators lead to the desired maxima distribution, but since we check all the possibilities, we will also come across these settings, where the intended maxima contribute as the maxima of μ_0.

Since we only fix the boundaries of the partitions that are intended to pose a maximum, we have to check, whether we can set up the other boundaries, such that the intended maxima remain the maxima. The following lemma shows, that we can check this in $O(N \cdot N_C)$.

Lemma 4.2 *Without loss of generality we assume that s sets the boundaries that uniquely define N_C different and not adjacent partitions that are intended to pose the maxima of μ_0, that is, if the partitions $P_{m_1}, \dots P_{m_C}$ are supposed to pose the maxima, then we assume, that the indices $s_{m_1}, s_{m_1+1}, s_{m_2}, s_{m_2+1}, \dots, s_{m_C}, s_{m_C+1}$ are set up. Then there is an $O(N \cdot N_C)$ algorithm, that checks in a constructive way whether the remaining boundaries can be set up such, that the intended maxima remain the actual maxima.*

Proof: First we compute the maxima m_{ω_c} for all c. This can be done in $O(N \cdot N_C)$. Then we consider the sets of adjacent indices that are not yet fixed, that is the sets $\{s_0, \dots, s_{m_1-1}\}, \{s_{m_1+2}, \dots, s_{m_2-1}\}, \dots, \{s_{m_C+2}, \dots, s_{N_P-1}\}$. For each of these sets we check whether the indices can be set such that the loads of the new partitions for each weight do not exceed the prior computed maxima m_{ω_c}. This can be done by traversing the sets from left to right defining partition boundaries such that none of the weights of these partitions barely exceed the maxima m_{ω_c}. If this procedure succeeds for all sets, we return *true*, otherwise *false*.

This constructive check requires at most $O(N \cdot N_C)$ steps, so the overall runtime is in $O(N \cdot N_C)$.

■

We use lemmata 4.1 and 4.2 to proof the existence of an algorithm that computes the optimal splitting of W into N_P parts.

[3] For boundary partitions there is only one separator to define, for two adjacent partitions there are only three boundary markers to set.

Theorem 4.3 *There is an algorithm that computes an optimal splitting s' of W into N_P parts in $O(N_P^{N_C} \cdot N^{2 \cdot N_C + 1} \cdot N_C)$.*

Proof:

Listing 4.1: Efficient algorithm for an optimal splitting of W into N_P parts

```
s' ← ⊥ (* stores a best solution *)                                  1
m ← ∞ (* stores μ_0(s') *)                                           2
                                                                     3
∀ maxima distributions P_{m_1}, P_{m_2}, ..., P_{m_{N_C}} do         4
   ∀ maxima partition boundary settings s do                         5
      if(extend s to a well−formed solution                          6
         as described in lemma 4.2)                                  7
         μ_t ← μ_0(s)                                                8
         if(μ_t < m)                                                 9
            m ← μ_t                                                  10
            s' ← s                                                   11
         end                                                         12
      end                                                            13
   end                                                               14
end                                                                  15
```

The correctness of the algorithm follows by construction. The outer loop is performed $O(N_P^{N_C})$ times, the inner loop is executed $O(N^{2 \cdot N_C})$. The body of the loop, that is the construction of a conform splitting from given boundaries for the maximum partitions, and the computation of $\omega_0(s)$ requires $O(N \cdot N_C)$ time. In total: $O(N_P^{N_C} \cdot N^{2 \cdot N_C} \cdot N \cdot N_C) = O(N_P^{N_C} \cdot N_C \cdot N^{2 \cdot N_C + 1})$.

■

4.2.4 Remarks

There are at least two reasons why we emphasize the existence of efficient algorithms for the above problem in such detail. First of all it is remarkable that we can efficiently (if the numbers of constraints are fixed) find a best solution to this problem as in the one-constraint situation. We do not apply any heuristics to find just good solutions, but we can compute in fact a best solution.

Secondly, this problem may also arise in different other settings, for instance directly as an array distribution problem. Suppose a one-dimensional array $A = [a_1, a_2, \ldots, a_n]$. Computations are performed on each a_i while there are data dependencies between adjacent elements of A, that is a_i is dependant on a_{i-1} and a_{i+1}. Suppose further that the computation of a_i requires time t_i and memory storage m_i. With the discussed algorithm, the matrix $\begin{pmatrix} t_1 & \ldots & t_n \\ m_1 & \ldots & m_n \end{pmatrix}$ can be split up in a locality preserving way that guarantees a simple communication pattern. This splitting balances the required computation time and the storage requirement under the constraint of only having communication among

adjacent processors. If the t_i and m_i vary only slightly the quality of the balancing and hence the parallel efficiency of the computation may be quite satisfactory.

Note that the aim for simple communication patterns are strongly demanded and this example is not too artificial. For instance the generalized block distribution splits a two dimensional array into rectilinear blocks such that the maximum sum of the elements of a single block is minimized. This way of decomposing an array guarantees a simple communication pattern. The problem of finding such a splitting is NP-complete, see [10], however heuristics are developed for approximating solutions efficiently, see [1].

4.3 Two constraint domain decomposition with space filling curves

This section describes our basic strategy of splitting in the two-constraint setting. In a first step the algorithm splits W into $\sigma \cdot N_P$ groups, $\sigma \in \{1, 2, \dots \lfloor \frac{N}{N_P} \rfloor\}$, and then it generates N_P partitions, each consisting of σ groups. We point out that the imbalance of the weights decreases with increasing σ. First we provide a case study to demonstrate and motivate the algorithm followed by a theoretical discussion.

4.3.1 Case study

We consider here an artificial two-constraint problem that does not correspond directly to a PIC decomposition problem. We denote the weights here with ω_1 and ω_2 and the weights are distributed in index space as illustrated in the topmost distribution in figure 4.1, ω_1 is colored green, ω_2 is colored blue. Even though the distribution of ω_1 is constructed in the same manner as in prior examples (e.g. 3.5), that is we take our habitual U-shaped geometry and set $\omega_1 = 1/N$ for all triangles, ω_2 does not correspond to a proper particle distribution. For particle simulations it holds that each particle is governed by a sourrounding triangle/tetrahedron, so if in index space ω_2 exhibits particles in some region, ω_1 also has to have some support in this region because it is constructed to indicate the appearances of triangles. This does not hold here and this fact makes the example artificial.

Figure 4.1 illustrates the decomposition phase of our algorithm, we decompose into $N_P = 3$ parts and equally set $\sigma = 3$.

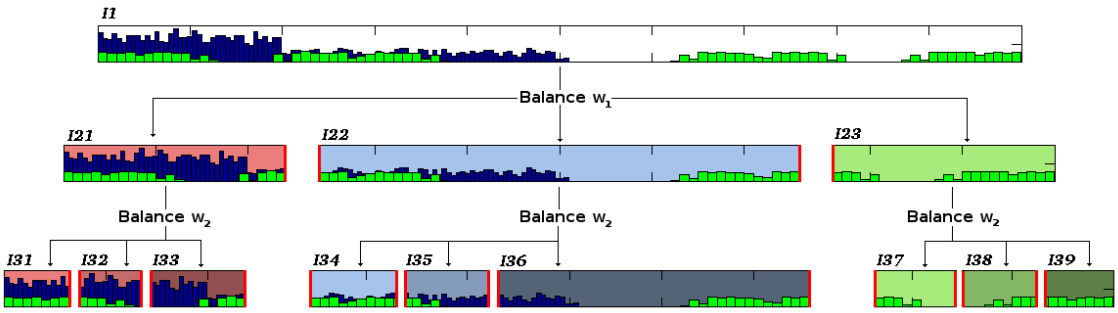


Figure 4.1: Decomposition phase of our basic algorithm for a two-constraint problem, $\sigma = N_P = 3$.

$I1$ displays the distribution of the two weights in index space, the unit interval $[0,1]$. In a first decomposition step we decompose $[0,1]$ into σ subintervals $I21, I22, I23$. This decomposition balances ω_1 in these intervals, such that

$$\omega_1(I21) = \omega_1(I22) = \omega_1(I23). \tag{4.9}$$

Balancing ω_1 is highly conflictive with balancing ω_2 in this example, $\omega_2(I23) = 0$ while $I21$ poses most of the contribution of ω_2.

The second step of the decomposition phase is to further decompose each of the σ subintervals $I21, I22, I23$ into $N_P = 3$ parts such that ω_2 is balanced for each new decomposition, hence:

$$\begin{aligned} \omega_2(I31) = \omega_2(I32) &= \omega_2(I33) \quad , \\ \omega_2(I34) = \omega_2(I35) &= \omega_2(I36) \quad , \\ \omega_2(I37) = \omega_2(I38) &= \omega_2(I39) \quad (= 0)\,. \end{aligned} \tag{4.10}$$

Note that any decomposition of $I23$ balances ω_2 simply because $\omega_2(I23) = 0$. Here we decomposed $I23$ such that ω_1 is balanced, but we could have chosen any other decomposition, for our considerations the only requirement is that ω_2 is balanced.

We make use of the properties 4.10 and 4.9 in the reunification step. By the equations 4.10 we observe that forming partitions by picking from each *set of groups*

$$S_1 = \{I31, I32, I33\}, S_2 = \{I34, I35, I36\}, S_3 = \{I37, I38, I39\} \tag{4.11}$$

one interval, leads to a balancing of weight ω_2 *automatically*. This allows us to select intervals such that ω_1 is balanced as good as possible. Figure 4.2 shows the best combination for our example, each path represents a final partition.

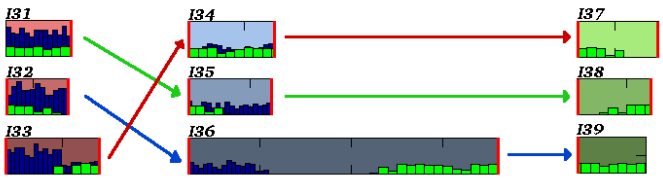


Figure 4.2: Reunification phase of our two-constraint algorithm. Each path represents a partition.

We formally define the reunification problem:

Problem 4.4 *The reunification problem (REUNIFICATION) consists of, given* $\sigma, N \geq 2$, *the sets* $A_1, \ldots A_\sigma \subset \mathbb{R}_0^+$ *with* $|A_i| = N \quad \forall i$, *find* $P_1, \ldots P_N \in A_1 \times \cdots \times A_\sigma$ *with* $P_i^k = P_j^k \Leftrightarrow i = j$ *such that:*

$$max\{\sum_{i=1}^{\sigma} P_k^i\} \rightarrow min. \tag{4.12}$$

In our concrete setting we have: $\sigma = N_P = 3$, $A_1 = \{\omega_1(I_{31}), \omega_1(I_{32}), \omega_1(I_{33})\}$, $A_2 = \{\omega_1(I_{34}), \omega_1(I_{35}), \omega_1(I_{36})\}$, and $A_3 = \{\omega_1(I_{37}), \omega_1(I_{38}), \omega_1(I_{39})\}$.

In the following we will denote the groups that are formed as a result of the second splitting operation (in the example $I31, \dots I39$) further on as *groups*, and we will use the character G with some index if we want to emphasize its representation as a part of matrix W. We will use the character I if we want to emphasize that we deal with intervals.

Furthermore, we will denote *sets of groups*, that contain these groups that are a product of the same second splitting operation, with the character S as we used it in 4.11. We refer to this structure as a *set of groups*.

REUNIFICATION is NP-complete, compare section 4.3.3, so for now we apply the following greedy heuristic (in section 4.3.4 we discuss a heuristic that performs better in average, the example in section 6.5 shows that it is absolutely necessary to apply the improved heuristic). As illustrated in figure 4.2 a solution to the reunification problem can be represented by connecting arrows between S_1, S_2, S_3. We start by connecting S_1 to S_2 in such a way, that the resulting (yet *incomplete*) partitions are balanced best. For doing this, we sort the intervals in S_1 with respect to ω_1 in increasing order, that is $I32, I33, I31$, and we sort the intervals in S_2 in decreasing order $I36, I34, I35$. Now we connect the items with the same position in the sorted lists, that is $[I32, I36], [I33, I34]$ and $[I31, I35]$. We continue by sorting these partitions in increasing order with respect to ω_1, and the partitions in S_3 in decreasing order with respect to ω_1. We unite them in the same manner as before, and we end up with $[I32, I36, I39], [I33, I34, I37]$ and $[I31, I35, I38]$. If $N_p > 3$ this procedure is repeated.

As already mentioned this procedure automatically balances ω_2. We argue in the following section 4.3.2 that this procedure produces decompositions d such that $\mu_0(\omega_1, d) < \frac{N_P+\sigma-1}{\sigma}$. For our setting ($N_P = \sigma = 3$) the balancing is guaranteed to be better than $\frac{5}{3}$, which would be a poor result but fortunately just happens for artificial cases. This balancing strategy applies quite well here, and we get $\mu_0(\omega_1, d) \approx 1.042$.

4.3.2 Quality of the balancing

We formalize the algorithm described in the case study in the previous section. Given is the matrix $(W \in \mathbb{R}_0^+)^{N \times 2}$ which will be decomposed into N_P parts and $\sigma \geq 2$. We summarize the decomposition algorithm:

- **Partition Phase:** Split W into σ parts $W = (S_1 | \dots | S_\sigma)$. It holds $\omega_1(S_i) = \omega_1(S_j) = \frac{1}{\sigma}, 1 \leq i, j \leq \sigma$.
  Further split each $(S_i)_{i=1\dots\sigma}$ into N_P parts, $W = (G_{11} | \dots | G_{1N_P} | \dots | \dots | G_{\sigma 1} | \dots | G_{\sigma N_P})$.
  It holds $\omega_2(G_{ki}) = \omega_2(G_{kj}), 1 \leq k \leq \sigma, 1 \leq i, j \leq N$.

- **Reunification Phase:** Form partitions $(P_i)_{i=1\dots N_P}$ via the following greedy algorithm:

Listing 4.2: REUNIFICATION HEURISTIC 1

```
P_i ={G_1i} ∀ i                                          1
                                                         2
for l = 2 ... σ                                          3
      sort P_i decreasingly with respect to ω_1          4
      sort W_li increasingly with respect to ω_1         5
      P_i ∪= W_li ∀ i                                    6
end                                                      7
```

.We denote d the by P_i induced decomposition.

REUNIFICATION HEURISTIC 1 (RH1) initializes each partition with one group of S_1. It traverses S_2 till S_σ and distributes the groups in these sets of groups to the partitions.

Proving the upper bound for $\mu_0(\omega_1, d)$ requires the following lemma.

Lemma 4.5 *Given $N \in \mathbb{N}$, $m \in \mathbb{R}^+$, $(a_i)_{i=1...N}, (b_i)_{i=1...N} \in \mathbb{R}_0^+$ with $a_i \leq a_{i+1}, 0 < i < N$ and $b_{i+1} \leq b_i, 0 < i < N$. Further it holds:*

$$\begin{aligned} max\{a_i\} - min\{a_i\} &= m_1, \\ max\{b_i\} - min\{b_i\} &= m_2. \end{aligned}$$

Then:

$$max\{a_i + b_i | 1 \leq i \leq N\} - min\{a_i + b_i | 1 \leq i \leq N\} \leq max\{m_1, m_2\}. \tag{4.13}$$

Proof: Proof by contradiction. Without loss of generality we assume $m_1 \geq m_2$. We assume that:

$$max\{a_i + b_i | 1 \leq i \leq N\} - min\{a_i + b_i | 1 \leq i \leq N\} > m_1. \tag{4.14}$$

i_{max}, i_{min} denote the indices that are responsible for the maximum and the minimum in 4.14. It holds further:

$$a_{i_{max}} + b_{i_{max}} - a_{i_{min}} - b_{i_{min}} > m_1 \quad \Leftrightarrow \tag{4.15}$$

$$\underbrace{a_{i_{max}} - a_{i_{min}}}_{A} + \underbrace{b_{i_{max}} - b_{i_{min}}}_{B} > m_1. \tag{4.16}$$

1. $i_{max} > i_{min} \Rightarrow 0 \leq A \leq m_1, -m_2 \leq B \leq 0 \Rightarrow A + B \leq m_1, \quad \lightning$
2. $i_{max} < i_{min} \Rightarrow -m_1 \leq A \leq 0, 0 \leq B \leq m_2 \Rightarrow A + B \leq m_1. \quad \lightning$

■

Theorem 4.6 $\mu_0(\omega_1, d) \leq \frac{N_P + \sigma - 1}{\sigma}$.

Proof: First we proof that it holds throughout the algorithm that $acc = max\{\omega_1(P_i)\} - min\{\omega_1(P_i)\} \leq \frac{1}{\sigma}$, hence the maximal imbalance is $\frac{1}{\sigma}$. We prove this by induction.
Induction Hypothesis: Since $\omega_1(W_1) = \frac{1}{\sigma}$ after the initialization in line 1 the hypothesis is obviously clear.
Induction Step: Following the same argument for the hypothesis, it is obvious that for $l > 1$ it holds that $acc_l = max\{\omega_1(W_{li})\} - min\{\omega_1(W_{li})\} \leq \frac{1}{\sigma}$. Applying Lemma 4.5 completes this first part of the proof.

□

We estimate $\omega_1(P_i) \quad \forall i$:

$$max\{\omega_1(P_i)\} - min\{\omega_1(P_i)\} \leq \frac{1}{\sigma} \quad \Leftrightarrow \tag{4.17}$$

$$max\{\omega_1(P_i)\} - \omega_1(P_i) \leq \frac{1}{\sigma} \quad \forall i \quad \Leftrightarrow \tag{4.18}$$

$$\omega_1(P_i) \geq max\{\omega_1(P_i)\} - \frac{1}{\sigma} \quad \forall i. \tag{4.19}$$

Further it holds:

$$1 = \sum_i \omega_1(P_i) \tag{4.20}$$

$$= max\{\omega_1(P_i)\} + \sum_{i \neq i_{max}} \omega_1(P_i) \tag{4.21}$$

$$\geq max\{\omega_1(P_i)\} + (N_P - 1)\left(max\{\omega_1(P_i)\} - \frac{1}{\sigma}\right) \tag{4.22}$$

$$\Leftrightarrow \tag{4.23}$$

$$\frac{N_P + \sigma - 1}{\sigma} \geq max\{\omega_1(P_i)\} \cdot N_P = \mu_O(\omega_1, d). \tag{4.24}$$

■

This theoretic bound tells us that by increasing σ the upper bound for the balancing decreases. However, increasing σ means increasing the number of subintervals per partition and hence the connectivity of the resulting partition decreases. This leads to higher edge-cuts. For this reason σ is a trade-off between balancing and edge-cut.

4.3.3 REUNIFICATION is NP-complete

The greedy algorithm presented in the prior section for the reunification problem guarantees the bound $\mu_0(\omega_1, d) \leq \frac{N_P+\sigma-1}{\sigma}$. This is indeed a sharp worst-case bound since we can construct examples where no algorithm can do better, see figure 4.3.

On the other hand, we can also construct examples where the provided reunification heuristic performs *worst*. Figure 4.4 shows an example that has a perfect balancing ($\mu_d(\omega_1, d_0) = 1$), however the reunification heuristic provides a solution d_h with $\mu_d(\omega_1, d_h) = \frac{N_P+\sigma-1}{\sigma}$.

The example of figure 4.4 demonstrates that there is an aim for improving the reunification step. However, since REUNIFICATION is NP-complete, which we prove in the following, there remains the application of better heuristics (see 4.3.4) that perform better in average.

We provide a reduction from the integer partitioning problem.

Problem 4.7 *Given is a set of N positive integers $M = \{m_1, \ldots, m_N\}$. The **partitioning problem** (PARTITION) consists of splitting M into disjoint sets M_1, M_2 such that $\sum_{m \in M_1} m - \sum_{m \in M_2} m$ is minimal.*

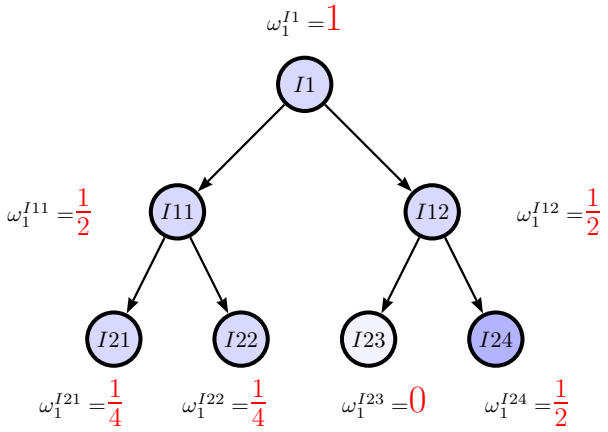


Figure 4.3: $\frac{N_P+\sigma-1}{\sigma}$ poses a sharp worst case bound for the balancing, the two possible reunifications $d_1 = \{[I21, I23], [I22, I24]\}$ and $d_2 = \{[I21, I24], [I22, I23]\}$ in this example ($\sigma = N_P = 2$) lead to $\mu_d(\omega_1, d_1) = 1.5 = \mu_d(\omega_1, d_2) = \frac{N_P+\sigma-1}{\sigma}$.

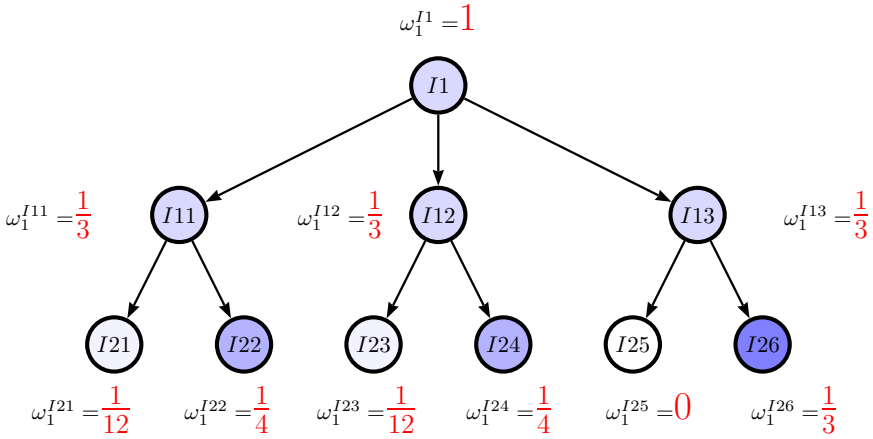


Figure 4.4: The reunification heuristic performs worst possible in this example ($N_P = 2, \sigma = 3$). It forms the reunification $d_h = \{[I21, I24, I26], [I22, I23, I25]\}$ with $\mu_d(\omega_1, d_h) = \frac{4}{3} = \frac{N_P+\sigma-1}{\sigma}$. Note that here exists a perfectly balanced decomposition $d_0 = \{[I21, I23, I26], [I22, I24, I25]\}$.

PARTITION is one of the famous 21 NP-complete problems of Karp, see [20], and serves as a reference problem for reductions of important NP-complete problems such as bin packing, multiprocessor scheduling or even the knapsack problem. Here we use this problem and reduce it to REUNIFICATION to show that REUNIFICATION is NP-complete. In section 4.3.4 we provide a better heuristic for REUNIFICATION that is inspired by the Karmarkar-Karp largest differencing algorithm, a heuristic for solving PARTITION. For this reason we will investigate on PARTITION in more detail in section 4.3.4.

Theorem 4.8 *REUNIFICATION is NP-complete.*

Proof: It is clear that REUNIFICATION $\in NP$ because the corresponding decision formulation of REUNIFICATION ("Are there $P_i = (P_i^1, \dots, P_i^\sigma)$ as a set of groups such that $\sum_l \omega_1(P_i^l) = m$ for a given m") has an efficient verifier.

It remains to show that REUNIFICATION is NP-hard. Therefore we reduce PARTITION to REUNIFICATION. We assume that we have an instance of PARTITION, that is the set $M = \{m_1, \dots, m_N\}$ of N integers. We construct now a REUNIFICATION problem that solves the PARTITION instance.

We set $N_P = 2, \sigma = N$. Furthermore we set $S_1 = \{m_1, 0\}, S_2 = \{m_2, 0\}, \dots S_\sigma = \{m_N, 0\}$. An algorithm for REUNIFICATION provides partitions P_1, P_2 that state the solution to the PARTITION instance.

■

4.3.4 A better heuristic for REUNIFICATION

We propose in this section a heuristic, REUNIFICATION HEURISTIC 2 (RH2), for REUNIFICATION that in average performs better than RH1. Note that, since RH1 is already optimal in terms of worst-case problems, the theoretic quality bound of $\frac{N_P+\sigma-1}{\sigma}$ can not be improved by any algorithm (not even exhaustive search). We do not provide an analysis of the average case behavior of RH2. Since RH2 is very similar to the Largest Differencing Method (LDM) of Karmarkar and Karp, compare subsection 4.3.4, the best known heuristic for the PARTITION problem, we point out that an analysis of RH2 may be inspired by an analysis for LDM.

REUNIFICATION HEURISTIC 2

The algorithm requires the definition of a diameter of a set of groups, and a diameter of data structures we call *pre-partitions* that appear when we *merge* sets of groups:

Definition 4.9 *The **diameter** of a set of groups S, $|S| = N \in \mathbb{N}$ with respect to a weight ω is defined as:*

$$diam(S) = max\{\omega(G) \quad | \quad G \in S\} - min\{\omega(G) \quad | \quad G \in S\}. \tag{4.25}$$

*The **diameter** of a pre-partition $\widetilde{P} = [[G_{11}, \dots, G_{1d}], \dots, [G_{N1}, \dots, G_{Nd}]]$ with respect to a weight ω, with $|\widetilde{P}| = N$ and $|\widetilde{P}_i| = d$, is defined as:*

$$diam(\widetilde{P}) = max\{\sum_{i=1}^{d} \omega(G_{ji}) \quad \forall j\} - min\{\sum_{i=1}^{d} \omega(G_{ji}) \quad \forall j\}. \tag{4.26}$$

For a pre-partition $\widetilde{P}$, $\widetilde{P}_{ij}$ indicates the jth element of the ith set of groups of $\widetilde{P}$, according to the labeling provided in the definition of the diameter of a pre-partition, this corresponds to group G_{ij}.

A pre-partition is hence an ordered set of $N \times d$ groups. By its inner bracketings a pre-partition keeps track of groups that will definitely go into the same partition in the end. It is an ordered set of sets of groups, each of the inner sets containing the same number of elements. We will use pre-partitions as $N_P \times d$ data structures to group N_P times d groups. The name *pre-partition* is chosen, since a pre-partition is not yet a partition, however, it already groups groups that will definitely go into the same partition.

RH2 *merges* pre-partitions via the following algorithm:

Listing 4.3: Merge operation

```
(* P~ pre-partition with dimensionality N_P × p *)
(* Q~ pre-partition with dimensionality N_P × q *)
(* R~ pre-partition with dimensionality N_P × (p+q) *)

function merge(P~, Q~)
        sort P~ in ascending order with respect to diam
        sort Q~ in descending order with respect to diam
        R~ = [[P~_11, ..., P~_1p, Q~_11, ..., Q~_1p], ..., [P~_NP1, ..., P~_NPp, Q~_NP1, ..., Q~_NPp]]
        return R~
end
```

Now we can state the improved heuristic for REUNIFICATION. The quantities correspond to the quantities given in the problem definition of REUNIFICATION in 4.4.

Listing 4.4: REUNIFICATION HEURISTIC 2

```
P~^i = [[S_i1], ..., [S_iNP]]   ∀i ∈ {1, ... σ}

while ∃ more than one P~^i
        find i, j such that diam(P~^i) ≥ diam(P~^k)  ∀k  and
        diam(P~^i) ≥ diam(P~^k)  ∀k ∈ {1, ... σ} \ {i}
        P~^i = merge(P~^i, P~^j)
        delete P~^j
end
```

RH2 initializes pre-partitions $\widetilde{P}^i$ such, that each pre-partition contains one set of groups S_i. It merges iteratively those pre-partitions that have the largest diameter. Hence, after $\sigma - 1$ merges one pre-partition remains. The inner sets of this pre-partition indicate the groups that go into the same partition.

Comparison: RH1 and RH2

We give a REUNIFICATION instance in table 4.1 that shows how RH1 and RH2 perform.

The strategy of RH2 can be verified in table 4.1. Since the diameter of the final group corresponds to the imbalance of weight ω_1, our goal is to minimize the final diameter. In

RH2	**RH1**
$\overset{\sim 1}{P} = \begin{pmatrix} 1 \\ 5 \\ 6 \end{pmatrix}, \overset{\sim 2}{P} = \begin{pmatrix} 11 \\ 7 \\ 13 \end{pmatrix}, \overset{\sim 3}{P} = \begin{pmatrix} 0 \\ 7 \\ 9 \end{pmatrix}, \overset{\sim 4}{P} = \begin{pmatrix} 2 \\ 8 \\ 12 \end{pmatrix}$ $dist(\overset{\sim 1}{P}) = 5, dist(\overset{\sim 2}{P}) = 6, dist(\overset{\sim 3}{P}) = 9, dist(\overset{\sim 4}{P}) = 10$	$P = \begin{pmatrix} 1 \\ 5 \\ 6 \end{pmatrix}, S_2 = \begin{pmatrix} 11 \\ 7 \\ 13 \end{pmatrix}, S_3 = \begin{pmatrix} 0 \\ 7 \\ 9 \end{pmatrix}, S_4 = \begin{pmatrix} 2 \\ 8 \\ 12 \end{pmatrix}$ $dist(P) = 5, dist(S_2) = 6, dist(S_3) = 9, dist(S_4) = 10$
Merge the two pre-partitions with maximal $dist$ $(\overset{\sim 3}{P}, \overset{\sim 4}{P})$: $\overset{\sim 1}{P} = \begin{pmatrix} 1 \\ 5 \\ 6 \end{pmatrix}, \overset{\sim 2}{P} = \begin{pmatrix} 11 \\ 7 \\ 13 \end{pmatrix}, \overset{\sim 3}{P} = \begin{pmatrix} 0 & 12 \\ 7 & 8 \\ 9 & 2 \end{pmatrix}$ $dist(\overset{\sim 1}{P}) = 5, dist(\overset{\sim 2}{P}) = 6, dist(\overset{\sim 3}{P}) = 4$	Merge next set of groups S_2 to P: $P = \begin{pmatrix} 1 & 13 \\ 5 & 11 \\ 6 & 7 \end{pmatrix}, S_3 = \begin{pmatrix} 0 \\ 7 \\ 9 \end{pmatrix}, S_4 = \begin{pmatrix} 2 \\ 8 \\ 12 \end{pmatrix}$ $dist(P) = 3, dist(S_3) = 9, dist(S_4) = 10$
Merge the two groups with maximal $dist$ $(\overset{\sim 1}{P}, \overset{\sim 2}{P})$: $\overset{\sim 1}{P} = \begin{pmatrix} 1 & 13 \\ 5 & 11 \\ 6 & 7 \end{pmatrix}, \overset{\sim 3}{P} = \begin{pmatrix} 0 & 12 \\ 7 & 8 \\ 9 & 2 \end{pmatrix}$ $dist(\overset{\sim 1}{P}) = 3, dist(\overset{\sim 3}{P}) = 4$	Merge next set of groups S_3 to P: $P = \begin{pmatrix} 1 & 13 & 7 \\ 5 & 11 & 0 \\ 6 & 7 & 9 \end{pmatrix}, S_4 = \begin{pmatrix} 2 \\ 8 \\ 12 \end{pmatrix}$ $dist(P) = 6, dist(S_4) = 10$
$\overset{\sim 1}{P} = \begin{pmatrix} 1 & 13 & 0 & 12 \\ 5 & 11 & 9 & 2 \\ 6 & 7 & 7 & 8 \end{pmatrix}$ $dist(\overset{\sim 1}{P}) = 2$	$P = \begin{pmatrix} 1 & 13 & 7 & 8 \\ 5 & 11 & 0 & 12 \\ 6 & 7 & 9 & 2 \end{pmatrix}$ $dist(P) = 5$

Table 4.1: Computation table of RH2 and RH1 for a REUNIFICATION instance. The entries in the pre-partitions and in the sets of groups represent here the weights of the groups.

each step RH2 tries to diminish the maximum of the diameters of the groups by one merge operation, hence picking the maximum element for a merge operation is necessary. In theorem 4.10 we show that the diameter of a set of groups S_3 that results as a merge from two sets of groups S_1, S_2 is smaller or equal than the larger diameter of S_1 and S_2 and can get as small as the difference between the diameters of S_1 and S_2. For this reason we also pick the second largest element for the merge operation, since the range of possible new diameters is $(dist(\text{largest element}) - dist(\text{second largest element})) \dots dist(\text{largest element})$ and hence the lower bound gets as small as possible by picking the second largest element.

One merge operation of pre-partition $\overset{\sim 1}{P}$ and $\overset{\sim 2}{P}$ hence corresponds to deciding which sets of groups of $\overset{\sim 1}{P}$ and $\overset{\sim 2}{P}$ go into the same partition. Note that the answer to into which partition they go is postponed to the final merge operation. For instance, considering $\overset{\sim 3}{P}$ after the second merge operation, it is decided that 0 and 12 go into the same partition (as well as 7 and 8, and 9 and 2), it is however not yet decided into which partition they will go.

Table 4.1 shows that the final partitioning P of RH1 *eats* the sets of groups from left to right without regards to the maximal diameter. The intention of RH1 is to assign in each step the elements of a set of groups directly and finally to a partition. This corresponds to immediately taking the decision into which partition a group will go. Compared to RH2 this early decision makes RH1 more static and inferior to RH2.

Basic Analysis

We only provide here a theorem concerning the diameter of a pre-partition that is the result of a merge operation.

Theorem 4.10 *Given* $S^1 \subset \mathbb{R}_0^{+d_1}$, $S^2 \subset \mathbb{R}_0^{+d_2}$, $|S^1| = |S^2| = N$. *We assume* $diam(S^1) > diam(S^2)$. *Then:*

$$diam(S^1) - diam(S^2) \leq diam(merge(S^1, S^2)) \leq diam(S^1). \tag{4.27}$$

Proof: We denote $S^3 = merge(S^1, S^2)$.

- $diam(S^1) - diam(S^2) \leq diam(S^3)$:
  Assume that S^1 is sorted in ascending order with respect to *diam*, S^2 is sorted in descending order with respect to *diam*. S^3 is the result of a merge operation of S^1 and S^2. It holds:

$$\sum_{i=1}^{d_1+d_2} S^3_{1i} = \sum_{i=1}^{d_1} S^1_{1i} + \sum_{i=1}^{d_2} S^2_{1i} \quad \text{and} \tag{4.28}$$

$$\sum_{i=1}^{d_1+d_2} S^3_{Ni} = \sum_{i=1}^{d_1} S^1_{Ni} + \sum_{i=1}^{d_2} S^2_{Ni}, \tag{4.29}$$

  and hence:

$$\begin{aligned} diam(S^3) &\geq |\sum_{i=1}^{d_1+d_2} S^3_{1i} - \sum_{i=1}^{d_1+d_2} S^3_{Ni}| && (4.30)\\ &= |\sum_{i=1}^{d_1} S^1_{1i} + \sum_{i=1}^{d_2} S^2_{1i} - \sum_{i=1}^{d_1} S^1_{Ni} - \sum_{i=1}^{d_2} S^2_{Ni}| && (4.31)\\ &= |(\sum_{i=1}^{d_1} S^1_{1i} - \sum_{i=1}^{d_1} S^1_{N,i}) + (\sum_{i=1}^{d_2} S^2_{1,i} - \sum_{i=1}^{d_2} S^2_{Ni})| && (4.32)\\ &= |-diam(S^1) + diam(S^2)| = diam(S^1) - diam(S^2). && (4.33) \end{aligned}$$

□

- $diam(merge(S^1, S^2)) \leq diam(S^1)$:
  Consequence of lemma 4.5.

□

■

Theorem 4.10 tells us that the diameter of two pre-partitions is bounded by the maximum of the diameters of the pre-partitions. Hence, the diameter of the final partitioning is bounded by the maximum diameter of a set of groups. This observation is already applied in the proof of the worst-case quality in 4.6.

Theorem 4.10, however, does not state the average decrease of the diameters as a result of a merge operation, which is a necessary quantity for a quality analysis of RH2.

Relation to the Karmarkar-Karp largest-differencing method

The LDM was introduced by Karmarkar and Karp in 1982 in [19]. This algorithm is the best known heuristic for PARTITION. Given is a set M of N positive numbers that is to be split into two disjoint subsets M_1, M_2 such that the difference of the sums of the subsets is minimal, that is $\sum_{m \in M_1} m - \sum_{m \in M_2} m \rightarrow \min$.

LDM starts with a sorted list of the N numbers. Each step it decides that the two largest elements l_1, l_2 in this list go into different partitions. It removes l_1 and l_2 from the list, computes their difference $|l_1 - l_2|$, and inserts this difference again into the list. $|l_1 - l_2|$ represents the imbalance of the two sets M_1 and M_2 that is caused by the decision that l_1 and l_2 go into different partitions. Note that, at this point, the algorithm does not decide into which partition l_1 and l_2 go, it only determines that l_1 and l_2 go into different partitions. By keeping track of the procedure we can determine bottom-up the subsets M_1 and M_2. This procedure is illustrated in table 4.2, which provides an example taken from [11].

set to be partitioned	difference	left	right
19, 17, 13, 9, 6	2	13, 19	17, 9, 6
13, 9, 6, 2	4	13, 2	9, 6
6, 4, 2	2	4, 2	6
2, 2	0	2	2
0			0

Table 4.2: Example of LDM for PARTITION.

RH2 performs similarly. RH2 starts with a sorted list of pre-partitions, each pre-partition is represented by its diameter, hence we have a sorted list of σ numbers. We merge the two pre-partitions with the largest diameters, and we replace these two pre-partitions by one pre-partition. In terms of diameters we replace the two largest diameters by a new diameter. And here is the difference to the LDM of Karmarkar and Karp: in their version two numbers are replaced by their difference, whereas we replace two diameters d_1, d_2 by a new diameter $\in [|d_1 - d_2|, max(d_1, d_2)]$. To underline the similarity of RH2 to LDM we provide in table 4.3 the evolution of the diameters of the example given in table 4.1 in the same manner as the evolution of the numbers of LDM.

diameters	difference	replaced by
10, 9, 6, 5	1	$4 \in [10 - 9, 10]$
6, 5, 4	1	$3 \in [6 - 5, 6]$
4, 3	1	$2 \in [4 - 3, 4]$
2		

Table 4.3: Evolution of the diameters of pre-partitions of the example of table 4.1

Analysis of LDM provides indication for Analysis of RH2

Recently (in 2008) Boetcher and Mertens provide in [4] an analysis of LDM. They assume N real random numbers in $[0,1]$ and prove that the expectation value of the final *discrepancy*, that is the difference of the sums of the elements of the two sets, is $N^{-\frac{lnN}{2ln2}}$.

The idea arises to adapt their proof for RH2. This requires at first to analyze the expectation value of the new diameter of a pre-partition that is the result of a merge operation. This already turns out to be rather complicated. Firstly, it is difficult to see to which extend the weights of the elements of (already merged) pre-partitions are independent, since the sequence of merge operations is not completely random. Secondly, it holds for the weights of the groups $G_{i1},\ldots,G_{i\sigma}$ that belong to one set of groups S_i that $\sum_{j=1}^{N_P}\omega(G_{ij})=\frac{1}{\sigma}$ (due to the fact stated in 4.9), and hence the weights are not even intially randomly chosen.

The property that is important in the proof of Boetcher and Mertens is that two numbers $a,b\in O(1)$ get replaced by a number in $O(1/N)$ ($N=|M|$), a property that follows from the random distribution of numbers they assume. It has to be clarified if this property can be assumed in our setting.

4.4 Influence of σ on the edge-cut

σ determines the number of intervals in index space a partition is made of. Since SFCs are locality preserving this means in general that a partition consists of σ disconnected subparts. Since multiple similarly shaped subparts covering a volume V have in total a larger surface than one up scaled part of the same shape that covers the same volume [4], we should keep σ as small as possible. Here we provide an estimation of the qualitative influence of σ to the surface sizes ($\sim$ edge-cut [5]) of a partition.

4.4.1 The surface to volume ratio

The surface to volume ratio of partitions induced by SFCs is matter of ongoing research. Zumbusch shows in [35] that the following equation holds for *many* SFC-induced partitions:

$$s \leq C_{\text{part}} \cdot v^{(d-1)/d}, \tag{4.34}$$

with s the surface of a partition induced by a SFC, v the volume of the partition, d the dimension of the curve and C_{part} a constant depending on the SFC. Hungershöfer and Wierum provide in [15] some results for worst-, average- and best-case surface to volume ratios which are normed to the surface to volume ratio of a square for some two dimensional SFCs . Based on 4.34 we provide a *back-of-the-envelope* calculation for estimating the increase of edge-cut with the increase of σ.

4.4.2 Back-of-the-envelope calculation

We assume we decompose a volume $V=[0,1]^d$ into N_P parts with our two-constraint algorithm. For this estimation we assume further that all subparts are of the same shape.

[4] For instance 2 cubes covering a volume with measure 1 have a surface of $12\cdot(\frac{1}{2})^{\frac{2}{3}}\approx 7.55$ while one cube with volume 1 has a surface of 6.

[5] We assume that the triangulation is such that the edge-cut is well represented by the surface sizes.

Since we construct $\sigma \cdot N_P$ subparts in total, a subpart has a volume of $\frac{1}{N_P \cdot \sigma}$. We estimate the surface in dependency of σ and the number of partitions N_P:

$$\begin{aligned}
\text{“surface of a partition”} &= \text{“number of its subparts”} \cdot \text{“surface of subpart”} \\
&= \sigma \cdot C_{\text{part}} \cdot \text{“volume of subpart”}^{(d-1)/d} \\
&= \sigma \cdot C_{\text{part}} \cdot \left(\frac{1}{N_P \cdot \sigma}\right)^{(d-1)/d} \\
&= C_{\text{part}} \cdot \left(\frac{1}{N_P}\right)^{(d-1)/d} \cdot \sigma^{\frac{1}{d}}, \qquad (4.35)
\end{aligned}$$

since we assume that the edge-cut $\sim$ surface sizes, we state:

$$\text{“edge-cut of a partition”} \in O\left(N_P^{\frac{1-d}{d}} \cdot \sigma^{\frac{1}{d}}\right). \qquad (4.36)$$

The overall edge-cut of the partitioning is hence:

$$\begin{aligned}
\text{“edge-cut of entire partitioning”} &= \frac{1}{2} \cdot N_P \cdot \text{“edge-cut for one partition”} \\
&\in O\left(N_P^{\frac{1}{d}} \cdot \sigma^{\frac{1}{d}}\right). \qquad (4.37)
\end{aligned}$$

Estimation 4.37 shows that in two dimensions the edge-cut grows with $\sqrt{\sigma}$ which is worse than the situation in three dimensions where the edge-cut grows with $\sqrt[3]{\sigma}$.

If we wish to balance ω_1 within some balancing ϵ, we can set up a relation between σ and N_P according to the worst-case quality bound of our algorithm (as provided in 4.3.2):

$$\epsilon \leq \frac{N_P + \sigma - 1}{\sigma} \quad \Rightarrow \qquad (4.38)$$

$$\sigma \leq \frac{N_P - 1}{\epsilon - 1} \in O(N_P). \qquad (4.39)$$

Applying this in 4.37 leads to:

$$\text{“edge-cut of entire partitioning”} \in O\left(N_P^{\frac{2}{d}}\right). \qquad (4.40)$$

4.4.3 Discussion

The calculation is very imprecise, since we make lots of assumptions (the shape and the size of subparts are all the same, we aim to divide into equally sized partitions and subparts, we assume that the triangulation is such that the surface sizes represent the edge-cut quite well ...) as it is the nature of back-of-the-envelope calculations. Nevertheless, the calculations provide insight into at least two interesting aspects.

Firstly, as equation 4.35 and 4.37 show, the surface of a partition and the surface of the entire partitioning grows with $\sigma^{\frac{1}{d}}$. Therefore, increasing σ has a stronger impact on the

edge-cut in 2D stronger than in 3D. Furthermore, this dependency shows that increasing a small σ (e.g. from 2 to 3) worsens the edge-cut more significant than increasing large σ (e.g. from 10 to 11). If we think about σ as the number of subparts of a partition, then an interpretation is that the step from connected partitions ($\sigma = 1$) to disconnected partitions ($\sigma = 2, 3, 4$) is *worse* than increasing the number of *patches* $\sigma = 10, 11, ...$ if we already have a patchwork.

We verify the quantitative increase of edge-cut with respect to σ and the dimensionality in the testcase in section 6.1.

Secondly, equation 4.40 shows the worst case growth rate of the edge-cut with respect to the number of partitions. Since $\sigma \in O(N_P)$ in worst-case, we get that the edge-cut of the entire partitioning is in $O(N_P^{\frac{2}{d}})$. Note that for a constant σ [6] this growth rate would be in $O(N_P^{\frac{1}{d}})$, which is a quasi-optimal growth rate [7].

4.5 Simplifying difficult two-constraint problems

Hitherto we discussed our algorithm for the two-constraint problem based on *some* SFC. In 4.1 we state that we sort the tetrahedra with respect to some SFC, but we do not further discuss the influence of the SFC on our algorithm. Since the SFC defines the weight distribution in index space, the SFC actually defines the input for our two-constraint algorithm.

Based on experiments with our algorithm we could identify weight distributions in index space that are difficult to balance for our algorithm. For these cases we apply new, problem-specific SFCs that produce distributions that are easier to balance for our algorithm.

In this section we want to emphasize that the difficulty of the decomposition problem depends on the SFC. We provide methods in 2D and 3D to produce new SFCs that lead to weight distributions that are easier to balance for our two-constraint algorithm.

4.5.1 Difficulty increases with heterogeneity

We consider problems as *difficult* if it requires to apply a huge σ to balance ω_1 (as discussed ω_2 is balanced automatically). As discussed in section 4.4, σ should be kept as small as possible to limit the edge-cut of the decomposition.

In the context of two-constraint particle simulations we observed that our two-constraint algorithm performs poorly if the particle distribution in index space is highly heterogeneous, that is for instance if there are regions with lots of particles and regions with only a few particles. The fact that the particle distribution is determined by the underlying SFC gives rise to the idea to apply a different SFC that produces more homogeneous distributions. One could even view the problem from rear: "*Given a two-constraint decomposition problem find a space filling curve such that our decomposition algorithm performs best*".

[6] We assume that σ is in $O(1)$ and hence independent from N.

[7] If we could decompose into d-dimensional spheres, which are known to have the optimal surface to volume ratio, the edge-cut of the entire partitioning is in $O(N_P \cdot$ "Surface of sphere"$)$. For spheres (also for cubes and even SFC-induced partitions according to 4.34) it holds: "Surface" $\in O($"volume"$^{\frac{d-1}{d}}) = O((\frac{1}{N_P})^{\frac{d-1}{d}})$. Hence, the edge-cut of the entire partitioning grows with $O(N_P \cdot (\frac{1}{N_P})^{\frac{d-1}{d}}) = O(N_P^{\frac{1}{d}})$.

In this section we point out the idea of creating *problem-specific* SFCs, curves that suit better for individual problems. These curves lead to a more homogeneous weight distribution in index space and hence pose easier decomposition problems for our two-constraint algorithm. Note that we do not claim that these curves are in some sense optimal, neither do they provide the armamentarium to ease any two-constraint problem. We rather advert to the idea of constructing SFCs that are problem-specific.

The construction of our problem-specific SFCs is technically complex. We base our constructions on existing SFCs (we apply Moore's version of the Hilbert curve, since we exhaust the fact that it forms a Hamilton cycle) and show that locality preservation in terms of Hölder continuity is inherited to our new curves. We discuss these technical details in section 5.2.

4.5.2 Example in 2D

We illustrate this procedure with the decomposition problem displayed in figure 4.5. The unit square is triangulated with 19977 triangles. The example is constructed such that each triangle in the bottom-left subsquare contains 20 particles, the other triangles contain between 1 to 5 particles, randomly chosen. In total the example contains 144117 particles.

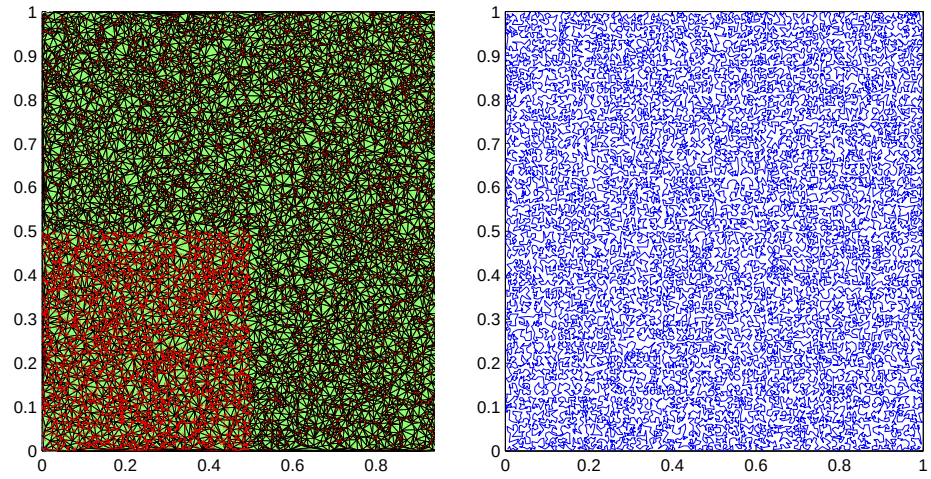


Figure 4.5: Left: Two-constraint decomposition problem in 2D: high particle density in the bottom-left subsquare. Right: Hilbert curve passing through the triangles.

As already discussed, the Hilbert curve in 2D visits all the points in the bottom left subsquare before it visits any point outside this subsquare. Hence most of the particles have an index between $[0, 0.25]$. The particle distribution is displayed in figure 4.6. $M = (0.25, 0.25)$ denotes the high density particle center.

We try to *ease* the problem by applying a SFC that traverses the unit square such that the indices of the particles are more scattered than in figure 4.6. Figure 4.7 shows a SFC, which we denote by $f_{a_4}^M$, that traverses the unit square such that it returns to the high density particle area from time to time.

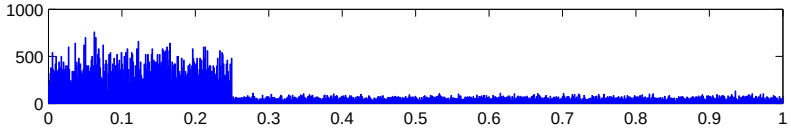


Figure 4.6: Particle distribution in index space of Hilbert curve.

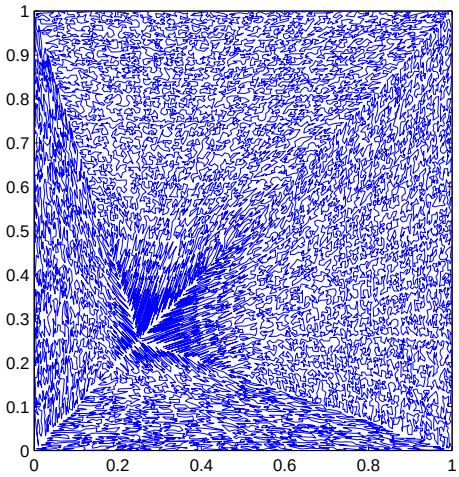


Figure 4.7: Problem-specific space filling curve $f_{a_4}^M$.

$f_{a_4}^M$ is constructed such that it starts at the center of the dense particles, hence $f_{a_4}^M(0) = M$. It traverses the entire triangle with the corner points $(0,0),(1,0),M$ and returns to point M. It further traverses the triangle $(0,0),M,(0,1)$ and again returns to point M. In this manner it passes through the triangle $(0,1),(1,1),M$, and finally the triangle $(1,1),M,(1,0)$, and returns to M such that $f_{a_4}^M(1) = M$. The intention of traversing the unit square *around a point* M with a high particle density to which we return a couple of times during the traversal is to scatter the high particle density in index space. Each return to M means that a fraction of the particles near M is visited (due to the locality of the underlying SFC). Hence, if we construct the curve such that the indices $I = (f_{a_4}^M)^{-1}(M)$ are well scattered over $[0,1]$, the dense particle center becomes more distributed in index space. Our method allows us to construct SFCs that visit the unit square around any point.

We loosely define:

Definition 4.11 *The two-dimensional problem-specific* SFC $f_{a_i}^P$, $P \in [0,1]^2, i \geq 4$ *denotes a* SFC *that traverses the unit square around point* P. *Therefore the unit square is decomposed into* i *triangles, and the triangles are visited in turn. The construction process is sketched in section 5.2.1.*

The distribution of the particles in index space of $f_{a_4}^M$ as displayed in figure 4.7 is illustrated in the upper plot of figure 4.8.

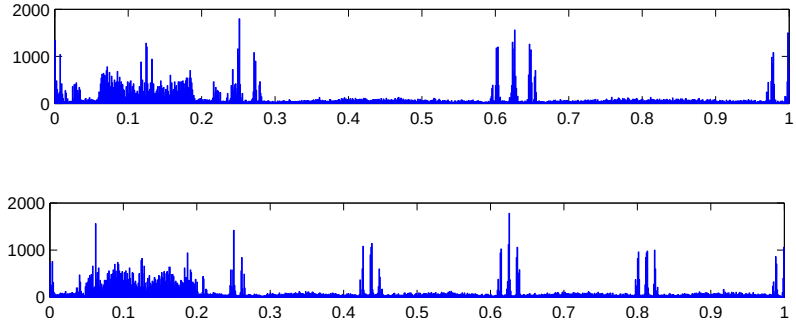


Figure 4.8: Particle distribution in index space of two problem-specific space filling curves, Top: a decomposition of the unit square into four triangles, ($f^M_{a_4}$), Bottom: a decomposition of the unit square into eight triangles ($f^M_{a_8}$).

The lower plot of figure 4.8 shows the index distribution with respect to a SFC that is also centered around M, but visits the unit square in a sequence of visits of 8 triangles.

Decomposing the domain into $N_P = 5$ parts with the choice of $\sigma = 3$ leads to an imbalance of 1.29 for ω_M, if our algorithm is based on the Hilbert curve. We achieve an imbalance of 1.04 for ω_M, if the algorithm is based on the problem-specific curve $f^M_{a_4}$. These decompositions are visualized in figure 4.9. In terms of edge-cut the decomposition based on the Hilbert curve achieves 652 cuts, while the decomposition based on the problem-specific curve achieves 690 cuts. The higher edge-cut of the decomposition based on the problem-specific curve is due to the fact that the adaptive curve is not as *local* as the Hilbert curve. We provide an estimation of the Hölder coefficient of the problem-specific curves in 5.2.2.

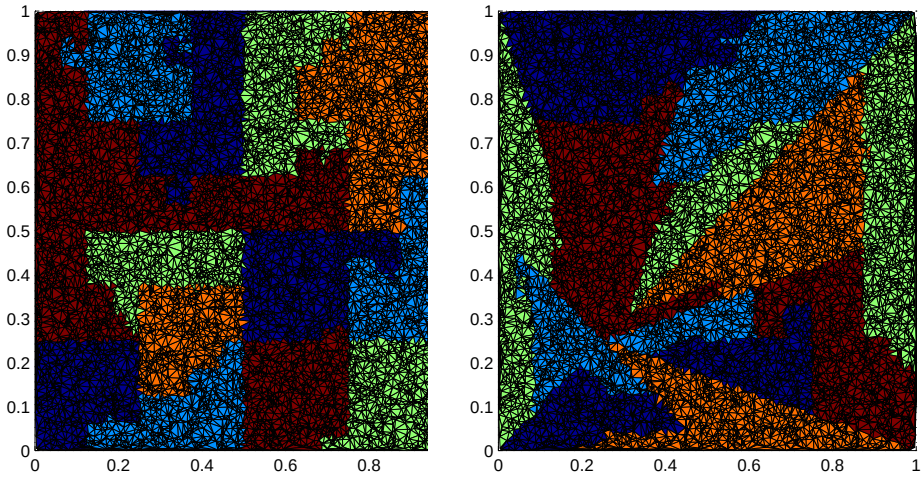


Figure 4.9: Two-constraint decompositions for the example displayed in figure 4.5, $\sigma = 3, N_P = 5$. Left: Hilbert curve, Right: Problem-specific curve.

4.5.3 Extension to 3D

The intuition of $f_{a_i}^M$ is extended to 3D. We denote the 3D versions of the problem-specific curves with $F_{a_i}^M$. Here we split the unit cube into at least six pyramids (the faces of the cube are the base areas of the pyramids, M serves as the apex) around a given point M. We construct SFCs that visit the unit cube as a sequence of visits to these six pyramids. Figure 4.10 shows a path that visits 3410 randomly picked points that are ordered with respect to the curve $F_{a_6}^{(0.5,0.5,0.5)}$. The path changes its color every $3410/6 \approx 568$ points. Since the points are chosen randomly but homogeneously, the 6 pyramids become visible.

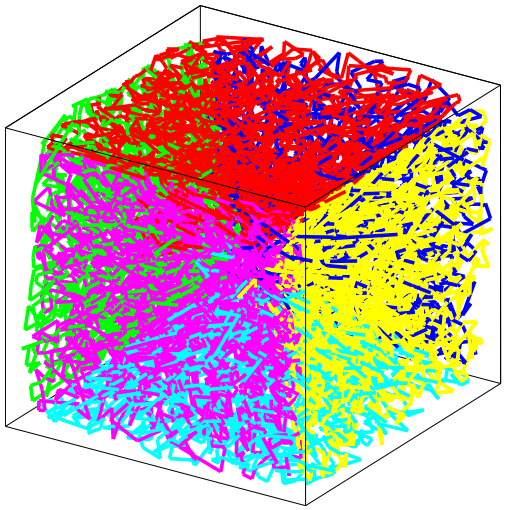


Figure 4.10: Space filling curve $F_{a_6}^{(0.5,0.5,0.5)}$ visiting the unit cube in a sequence of visits to 6 pyramids.

Definition 4.12 *The three-dimensional problem-specific* SFC $F_{a_i}^P$, $P \in [0,1]^3, i \geq 6$ *denotes a* SFC *that traverses the unit cube around point P. Therefore the unit cube is decomposed into i pyramids, and the pyramids are visited in turn.*

4.6 Mixed-constraint problems

We regard problems with $\omega_P(\tau) = 0$ for many τ as *mixed-constraint* problems. The problem provided in 3.2.5 poses a mixed-constraint problem since all the particles are localized in a small area top right, and for many triangles τ it holds $\omega_P(\tau) = 0$. These problems can be dealt with using our multi-constraint algorithm. However, here we propose a method that leads to better edge-cut results in many cases.

4.6.1 Mixed-constraint algorithm

We assume an underlying SFC f that forms a Hamilton cycle, that is $f(0) = f(1)$, for instance any of our problem-specific curves or simply Moore's version of the Hilbert curve. We search the largest $Z \subset [0,1]$ such that $\forall \tau$ with $f^{-1}(cg(\tau)) \in Z$ it holds that $\omega_P(\tau) = 0$.

Z is either an interval or of the form $Z = Z_1 \dot\cup Z_2$ with $Z_1 = [0, z_1]$ and $Z_2 = [z_2, 1]$, $0 < z_1 < z_2 < 1$. The Hamilton cycle allows us to obtain two index sets Z and $N = [0, 1] \setminus Z$ such that:

1. f is continuous on Z and on N,
2. $\omega_P(Z) = 0$, and
3. $\omega_P(N) > 0$.

We first decompose the set N with our two-constraint algorithm into N_P parts with a small σ which may lead to some imbalance. Now the set Z serves to compensate for the imbalance. Since $\omega_P(Z) = 0$, decomposing Z corresponds to a single-constraint problem into which the arisen imbalance of the decomposition of N is incorporated. We formulate this as follows:

Listing 4.5: mixed-constraint algorithm

```
decompose N via our two-constraint algorithm into (N_1,... N_{N_P})

decompose Z into Z_1,... Z_{N_P} such that:
    if ∄ N_i : ω_M(N_i) > 1/N_P
                ω_M(Z_i) = 1/N_P − ω_M(N_i)   ∀i
    else
                ω_M(Z_i) + ω_M(N_i) ≤ max {ω_M(N_j)}   ∀i
    end if

P_i = N_i ∪ Z_i   ∀i
```

To obtain a simple notation, the formulation of the algorithm is based on intervals. Here, a partition is the union of subintervals of $[0, 1]$.

In line 1 we apply our two-constraint algorithm to divide the interval N into disjoint subintervals (if $\sigma = 1$), or disjoint unions of subintervals ($\sigma > 1$).

In line 4 we check whether we can compensate the imbalance completely by the second decomposition, that is the case if there is no N_i such that its weight $\omega_M(N_i)$ already exceeds $\frac{1}{N_P}$ (since $\omega_M([0, 1]) = 1$). If this is the case, we compute the weights of Z_i in line 5. The actual decomposition then can be done by traversing the interval Z from left to right, accumulating ω_M and setting split points appropriately, as described in 3.2.4.

If $\omega_M(Z)$ is not sufficient to adjust the imbalances of the first decomposition, it only remains to split Z such that the total weight of a final partition does not exceed the largest $\omega_M(N_i)$. Note that there are multiple solutions. However, they are all equivalent in terms of μ_w or idletime as defined in 2.6.

4.6.2 Example

We base the following example on Moore's version of the Hilbert curve.

As displayed in figure 4.11, we consider a U-shaped geometry with particles restricted to the top right area. The particle distribution in index space is limited to a very small interval

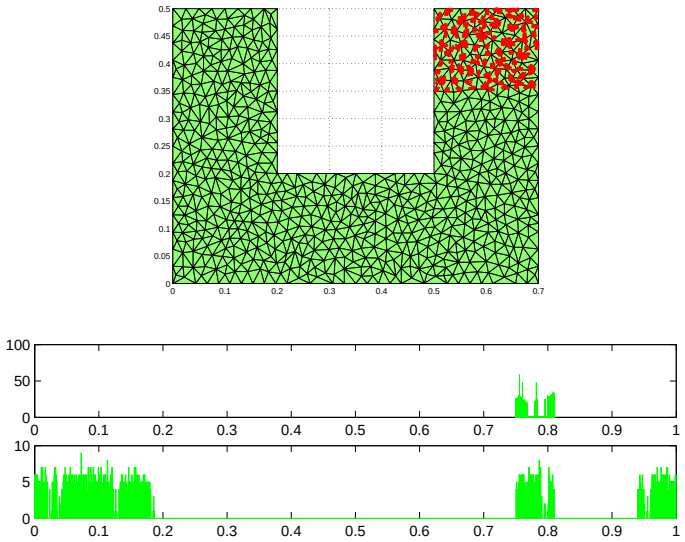


Figure 4.11: Mixed-constraint decomposition problem: particles are restricted to the top-right area. The upper histogram displays the particle distribution in index space. The lower histogram displays the distribution of the triangles in index space.

$N = [0.750195, 0.809903]$ and hence $Z = [0, 0.750195) \cup (0.809903, 1]$. Here we decompose into $N_P = 5$ parts and set $\sigma = 1$. Hence, the mixed-constraint algorithm corresponds to two single-constraint decompositions. The result is displayed in figure 4.12.

First N is decomposed such that each partition contains the same number of particles. We can see that a certain imbalance arises: the subparts of the orange and the brown partition with indices in N consist of much more triangles than the dark blue, the light blue, and the green partition. This is considered in the decomposition of Z. We can see, for instance, that the subpart of the brown partition with indices in Z is much smaller than the subpart of the green partition with indices in Z, so that in total the brown and the green partition contain the same number of triangles.

Note that the subpart of the indices of the light blue partition in Z is disconnected ($[0, 0.05]$ and $[0.985, 1]$). Due to the Hamilton cycle of Moore's curve the corresponding subpartition is connected.

4.6.3 Remarks

The provided example is chosen such that there are enough triangles without particles to equilibrate in the second decomposition the imbalance that occurred in the first decomposition, even though σ is chosen to be 1, which in general leads to the highest imbalance. If, however, the particles are more scattered, such that there are fewer triangles without particles, the situation may occur that it is not possible to adjust the imbalance of the first decomposition. In these cases σ can be increased. Note that if there are only few triangles without particles (more precisely the triangles that have an index in Z), we may achieve better decomposition results, if we treat the problem as a two-constraint problem, not as a mixed-constraint problem.

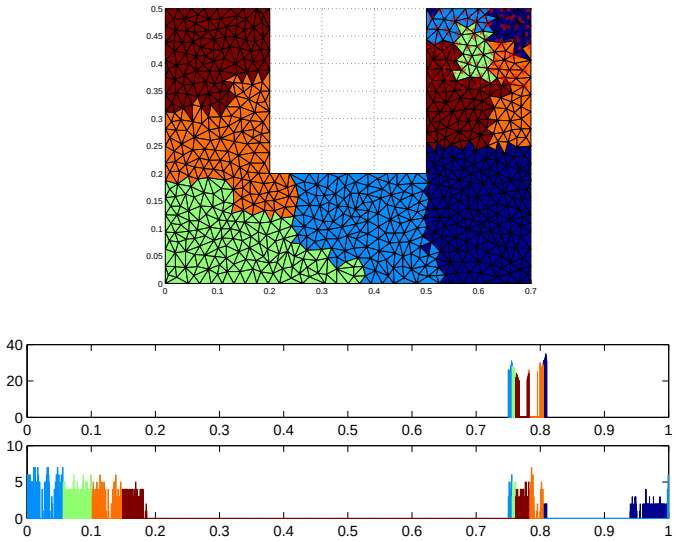


Figure 4.12: Result of mixed-constraint decomposition problem: in a first step the area that governs the particles is decomposed. In a second decomposition the remaining area is decomposed under consideration of the imbalances that arose during the first decomposition. The upper histogram displays the particle distribution in index space, the lower histogram displays the triangle distribution in index space.

Note, furthermore, that the *difficulty* here depends on the underlying SFCas well, however in a manner contrary to the one described in section 4.5. While for the two-constraint algorithm the particles should be scattered in index space as homogeneously as possible, here it is beneficial if particles are concentrated in a small interval (and as homogeneously concentrated in this interval as possible). Optimally a SFC traverses the unit cube/square in such a way that it hits all triangles that contain particles in a sequence and all triangles without particles in a sequence. In this setting the number of triangles for the second decomposition is maximal, hence, the potential to equilibrate imbalances of the first decomposition is maximal.

5 Technical details

This chapter provides insight in technical aspects of this work. Beside an explanation of how we actually compute the inverse mappings of SFCs we state the construction of our problem-specific SFCs. We present a proof scheme for Hölder continuity of these curves.

5.1 Computing the inverse of Space Filling Curves

The present report makes use of four SFCs, the Hilbert curve in 2D and in 3D and Moore's version of the Hilbert curve in 2D and in 3D. For the computation of the Hilbert mappings in any dimension Butz presented in [5] an algorithm, known as the Butz-algorithm, that can be easily implemented. Lawder presented in [24] how the Butz-algorithm can be *inverted* to compute the inverse of the Hilbert mapping.

In [18] Guohua and Mellor-Crummey present *SFCGen*, a framework for computing mappings of SFCs by recursion. The recursive scheme of a SFC is defined by tables which are sufficient to compute the mappings. For Moore's version of the Hilbert curve we developed an automaton-based algorithm which seems to be identical to the implementation scheme of *SFCGen*. This automaton-based scheme can also be applied for the Hilbert mapping, and it turned out that this scheme computes much faster than Butz algorithm, which meets the results of *SFCGen* provided in [18]. We explain the automaton-based scheme on the example of the Hilbert curve in 2D. For Moore's version of the Hilbert curve in 2D we provide the grammar that is sufficient to construct the curve. For Moore's curve in 3D we demonstrate that setting up the definition of the automaton requires cumbersome engineering work. For the Hilbert curve in 3D, therefore, we sticked to the inverse Butz algorithm to avoid setting up the automaton. Since we can compute inverse SFC mappings as a preprocessing step, efficiency is not crucial here.

5.1.1 Hilbert Curve in 2D

As in section 3.2.2, we compute the quarternary representation of the index $i = (0.i_1 i_2 \dots)_4$ of a two dimensional coordinate. For the coordinates a representation in dual system is convenient:

$$x = (0.x_1 x_2 \dots)_2 \tag{5.1}$$
$$y = (0.y_1 y_2 \dots)_2, \tag{5.2}$$

$x_i, y_i \in \{0, 1\}$. In each step i the automaton reads the tuple (x_i, y_i). And according to these values and the automaton's current state the automaton performs a transition into a new state and outputs a digit of the index. For the Hilbert curve in 2D the automaton in figure 5.1 computes the inverse mapping.

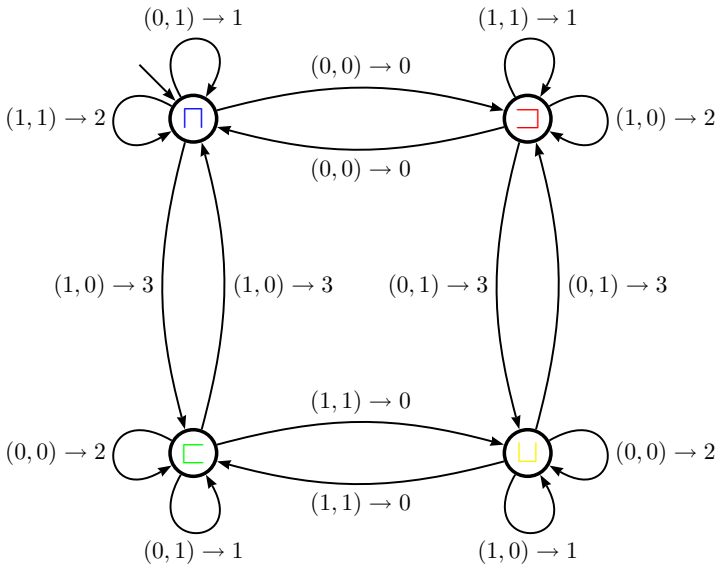


Figure 5.1: Automaton for computing the inverse 2D Hilbert function.

As an example we discuss the index computation to the point

$$(x, y) = (\frac{13}{16}, \frac{7}{16}) = ((0.1101)_4, (0.0111)_4)$$

that also demonstrates the functionality of the automaton and verifies its correctness.

The automaton starts in state ⊓. The first input are the first digits of the quarternary representation of (x, y), that is $(1, 0)$. Note that this indicates that the point lies in the bottom right subsquare. Therefore, the corresponding index is greater or equal to $0.75 = (0.3)_4$, which means that the first digit of the quarternary representation of the index is 3. The automaton indicates this by the transition from state ⊓ to ⊏, which is labeled by $(1, 0) \to 3$.

The next input tuple consists of the second digits of the quarternary representation of (x, y), that is $(1, 1)$. The automaton indicates the transition from ⊏ to ⊔, labeled with $(1, 1) \to 0$, hence the second digit of the index is 0. This is also what we expect since the Hilbert curve enters the bottom right subsquare from the right half of the top right subsquare.

The third and fourth digits of the quarternary representation of (x, y) indicate further transitions from ⊔ to ⊐ and from ⊐ to ⊐, and the index computes to 0.3031. Note that this is not the final index of (x, y). The following inputs to the automaton are all $(0, 0)$ (since $((0.1101)_4, (0.0111)_4) = ((0.1101000\ldots)_4, (0.0111000\ldots)_4)$) and hence the computation continues. After at most 4 further transitions of the automaton the computation loops, and the quarternary representation of the index gets periodic. For this reason we chose a precision in advance which states the number of transitions the automaton performs for a computation. Indeed preselecting a desired precision p corresponds to a truncation of the spatial coordinate to a corner point of the pth approximating polygon of the Hilbert curve. Further details about approximating polygons can be found for instance in [2].

Note that the automaton can also be used to compute the Hilbert mapping itself. For this purpose we only have to turn around the arrows of the labels of a transition. Each step the automaton reads a digit of the quarternary representation of an index and produces a tuple of binary digits of the x and y coordinate of the corresponding point.

The construction of the automaton is completely determined by the grammar of the 2D Hilbert curve which is shown in table 3.1. The non-terminals (or the basic shapes) in the grammar pose the states of the automaton. For each occurring basic shape on the right side of a grammar rule we construct a transition from the state represented by the basic shape on the left side to the basic shape on the right side. Therefore, we get four transitions for each state of the automaton.

For setting up the labels for a transition, we observe the way a basic shape is passed through by the Hilbert curve. We explain this procedure by the example of the grammar rule:

$$\sqsupset \Leftarrow \sqcap \rightarrow \sqsupset \uparrow \sqsupset \leftarrow \sqcup$$

taken from 3.1. The Hilbert curve enters $\sqsupset$ at the bottom left subsquare, so we create the label $(0,0) \rightarrow 0$. $(0,0)$ indicates the bottom left subsquare, and the 0 at the right side of the arrow indicates the *time* when the corresponding subsquare is passed through. Next the curve enters subsquare $(1,0)$ (bottom right). We create the label $(1,0) \rightarrow 1$. In this manner we can construct labels for all transitions.

5.1.2 Moore's curve in 2D

Moore's version of the Hilbert curve (which we denote with f_m in 2D) forms a Hamilton cycle, that is the curve is closed ($\Rightarrow f_m(1) = f_m(0) = (0.5, 0)$).

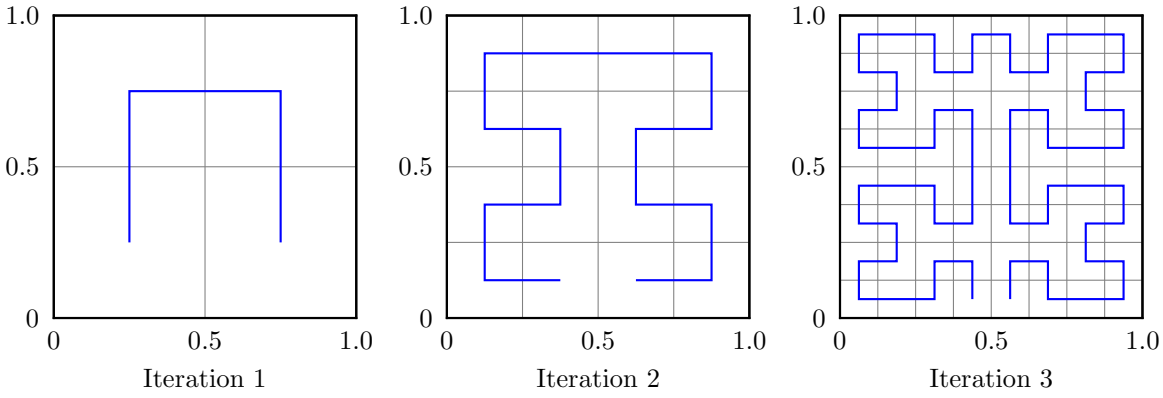


Figure 5.2: Iterations of Moore's version of the two dimensional Hilbert curve.

We assume that the curve starts in the bottom left subsquare and ends in the bottom right subsquare.

Five basic shapes are required for the grammar of Moore's curve. The basic shape $\sqcap$ poses two non terminals in the grammar since it is passed through both from bottom left to bottom right which we denote by $\sqcap$, and from bottom right to bottom left which we denote by $\sqcap$'. A grammar can be formulated as follows:

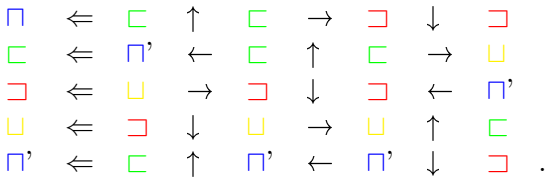


Table 5.1: Grammar for Moore's version of the 2D Hilbert curve.

The grammar in table 5.1 shows that the initial basic shape (⊓) does not appear on the right sides of any production. For this reason the automaton does not return to the initial state but only performs transitions among the other four basic shapes in the same way the automaton of Hilbert's curve does. For this reason Moore's curve can be seen as a Hilbert curve with an altered initial configuration.

The corresponding automaton is constructed in the same way as the automaton to the two-dimensional Hilbert curve.

5.1.3 Moore's curve in 3D

Constructing a grammar which defines an automaton for Moore's curve in 3D turned out to be cumbersome engineering work, since there was no literature found that provides a grammar for Moore's curve in 3D. While the two dimensional Hilbert curve is unique (except for symmetry), in 3D there are multiple Hilbert curves (see [2]), and hence multiple versions of Moore's curve. Since we only need one version of Moore's curve, we follow the following strategy to set up a grammar: we start with basic shape one (compare with table 5.2) and define a *reasonable* substitution rule for it. *Reasonable* here means that the sequence of basic shapes and connections form a continuous curve, and as far as possible we try to keep symmetry during this construction. For all basic shapes on the right sides of grammar rules we define substitutions. This procedure ends when we have defined a substitution rule for all appearing basic shapes. Our approach leads to the grammar displayed in table 5.3.

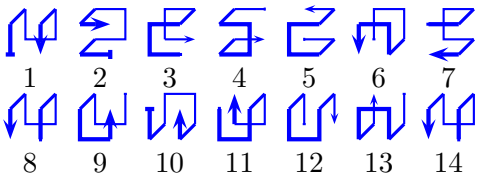


Table 5.2: Basic shapes for Moore's version of the 3D Hilbert curve.

Note that in three dimensions we substitute a basic shape by eight smaller basic shapes. Due to the third dimension we have two further connecting arrows (↗, ↙).

Similarly to the grammar for Moore's curve in 2D the initial basic shape does not appear on any right side of a production. In addition, all grammar rules, except the first one, exhibit symmetry and follow the structure $A \Leftarrow B, C, C, D, D, E, E, F$. This coincides with the grammar for Moore's curve in 2D.

1	⇐	2	↑	3	↗	3	↓	4	→	5	↑	6	↙	6	↓	7
2	⇐	8	←	3	↗	3	→	5	↑	5	←	9	↙	9	→	10
3	⇐	8	←	2	↑	2	→	11	↗	11	←	4	↓	4	→	12
4	⇐	13	←	9	↙	9	→	7	↓	7	←	3	↗	3	→	12
5	⇐	12	→	6	↙	6	←	2	↑	2	→	11	↗	11	←	13
6	⇐	5	↑	12	→	12	↓	9	↙	9	↑	14	←	4	↓	7
7	⇐	10	→	11	↗	11	←	4	↓	4	→	12	↙	6	←	14
8	⇐	2	↑	3	↗	3	↓	13	←	13	↑	6	↙	6	↓	7
9	⇐	4	↓	13	←	13	↑	6	↙	6	↓	10	→	10	↑	2
10	⇐	7	↓	11	↘	11	↑	12	→	12	↓	9	↖	9	↑	2
11	⇐	7	↓	10	→	10	↑	3	↗	3	↓	13	←	13	↑	5
12	⇐	5	↑	6	↙	6	↓	10	→	10	↑	3	↗	3	↓	4
13	⇐	4	↓	9	↙	9	↑	14	←	14	↓	11	↗	11	↑	5
14	⇐	2	↑	3	↗	3	↓	13	←	13	↑	6	↙	6	↓	7

Table 5.3: Grammar for Moore's version of the 3D Hilbert curve.

The construction of the corresponding automaton is analogous to the 2D case. In three dimensions a label is of the form $(x, y, z) \rightarrow \{0, \dots 7\}$. The computed index is provided in base eight system, since one basic shape passes through eight subcubes in 3D.

5.2 Construction of composite Space Filling Curves

In section 4.5 we discuss the use of problem-specific SFCs, $f_{a_i}^M$ in 2D and $F_{a_j}^N$ in 3D for points M, N and integers i, j, that suit better for specific decomposition problems. Here we provide the construction of $f_{a_4}^M$. The scheme can be straight-forwardly extended to the 3D case.

Our constructions are based on Moore's version of Hilbert's curve, which we denote by f_m in 2D. Now we discuss the curve $f_{a_4}^M$, a SFC that visits the unit square around point $M \in (0,1)^2$ as a sequence of visits to 4 triangles.

5.2.1 Definition of $f_{a_4}^M$

Figure 5.3 shows the course of the curve $f_{a_4}^M$:

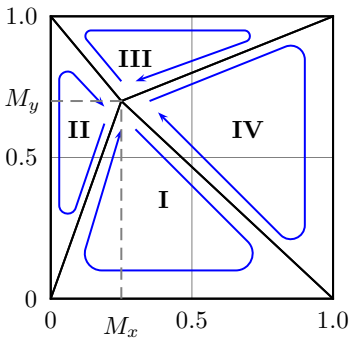


Figure 5.3: $f_{a_4}^{0.25,0.7}$ consecutively visits 4 triangles.

We denote the point sets of the triangles I, II, III, IV with $\triangle_I, \triangle_{II}, \triangle_{III}$ and $\triangle_{IV}$.

The curve starts at point M, visits triangle I, and returns to point M. In the same manner triangles II, III, and IV are visited. Since we aim that our problem-specific curves *evolve with constant speed*, triangle I covers the index interval $I_I = [0, A_I]$ with A_I the area of triangle I. For the same reason triangle II covers the index interval $I_{II} = (A_I, A_I + A_{II}]$, triangle III covers $I_{III} = (A_I + A_{II}, A_I + A_{II} + A_{III}]$, and triangle IV covers $I_{IV} = (A_I + A_{II} + A_{III}, 1]$ with A_i the area of triangle i.

For our construction we require contractions from the unit square to the triangles I, II, III, IV. These contractions map the point $(0.5, 0)$ to M. We assume that these mappings are Lipschitz-continuous (to a Lipschitz constant ≤ 1) and we denote as follows:

$$m_i : [0,1]^2 \rightarrow \triangle_i \tag{5.3}$$

$$||m_i(x_1) - m_i(x_2)||_2 \leq K_i \cdot ||x_1 - x_2||_2 \quad \forall i \in \{I, II, III, IV\} \text{ and } K_i \leq 1. \tag{5.4}$$

These mappings allow the definition of $f_{a_4}^M$:

$$f_{a_4}^M : [0,1] \rightarrow [0,1]^2 \tag{5.5}$$

$$f_{a_4}^M(i) = \begin{cases} m_1(f_m(\frac{1}{A_1} \cdot i)) & , i \in I_I \\ m_2(f_m(\frac{1}{A_2} \cdot (i - A_1))) & , i \in I_{II} \\ m_3(f_m(\frac{1}{A_3} \cdot (i - A_1 - A_2))) & , i \in I_{III} \\ m_4(f_m(\frac{1}{A_4} \cdot (i - A_1 - A_2 - A_3))) & , i \in I_{IV} \end{cases} \tag{5.6}$$

$$= m_j(f_m(\frac{1}{A_j} \cdot (i - \sum_{l<j} A_l))) \quad \text{for} \quad i \in I_j. \tag{5.7}$$

For a given index $i \in I_j$ the index range I_j is stretched to $[0,1]$. Moore's SFC mapping is applied, and the resulting point in $[0,1]^2$ is mapped to the triangle $\triangle_j$. Note that since $f_m(0) = f_m(1)$, $f_{a_4}^M$ is continuous. Further it holds that $f_{a_4}^M(0) = f_{a_4}^M(A_1) = f_{a_4}^M(A_1+A_2) = f_{a_4}^M(A_1+A_2+A_3) = f_{a_4}^M(1) = M$.

We show that the Hölder continuity of f_m is inherited to $f_{a_4}^M$:

5.2.2 Proof: $f_{a_4}^M$ is Hölder continuous

Theorem 5.1 *$f_{a_4}^M$ is Hölder continuous ($\forall M \in (0,1)^2$).*

Proof: C_m denotes the Hölder coefficient of Moore's curve. Given are further two indices $i_1, i_2 \in [0,1]$. We distinguish two cases:

1. $i_1, i_2 \in I_j$ for some j. We assume $i_1 > i_2$.

$$
\begin{aligned}
||f_{a_4}^M(\ \ i_1\ \ ) - f_{a_4}^M(i_2)||_2 = & && (5.8)\\
= & \ ||m_j(f_m(\frac{1}{A_j}\cdot(i_1 - \sum_{l<j} A_l))) - m_j(f_m(\frac{1}{A_j}\cdot(i_2 - \sum_{l<j} A_l)))||_2 && (5.9)\\
\leq & \ K_j \cdot ||f_m(\frac{1}{A_j}\cdot(i_1 - \sum_{l<j} A_l)) - f_m(\frac{1}{A_j}\cdot(i_2 - \sum_{l<j} A_l))||_2 && (5.10)\\
\leq & \ K_j \cdot C_m \cdot |\frac{1}{A_j}\cdot(i_1 - \sum_{l<j} A_l) - \frac{1}{A_j}\cdot(i_2 - \sum_{l<j} A_l)|^{\frac{1}{2}} && (5.11)\\
= & \ \frac{K_j \cdot C_m}{\sqrt{A_j}} \cdot |i_1 - i_2|^{\frac{1}{2}}. && (5.12)
\end{aligned}
$$

2. $i_1 \in I_j =: [j_1, j_2]\,, i_2 \in I_k =: [k_1, k_2]\,, j \neq k$. We assume that $i_1 > i_2$.

$$
\begin{aligned}
||f_{a_4}^M(i_1) - f_{a_4}^M(i_2)||_2 &= ||f_{a_4}^M(i_1) - f_{a_4}^M(j_1) + f_{a_4}^M(k_2) - f_{a_4}^M(i_2)||_2 && (5.13)\\
&\leq ||f_{a_4}^M(i_1) - f_{a_4}^M(j_1)||_2 + ||f_{a_4}^M(k_2) - f_{a_4}^M(i_2)||_2 && (5.14)\\
&\leq \frac{K_j \cdot C_m}{\sqrt{A_j}} \cdot |i_1 - j_1|^{\frac{1}{2}} + \frac{K_k \cdot C_m}{\sqrt{A_k}} \cdot |k_2 - i_2|^{\frac{1}{2}} && (5.15)\\
&\leq C_m \cdot \max\{\frac{K_j}{\sqrt{A_j}}, \frac{K_k}{\sqrt{A_k}}\} \cdot 2 \cdot |i_1 - i_2|^{\frac{1}{2}}. && (5.16)
\end{aligned}
$$

■

5.2.3 Inverting $f_{a_4}^M$ and practical issues

In practice we require the inverse mapping of $f_{a_4}^M$:

$$(f_{a_4}^M)^{-1}(x) = \begin{cases} A_1 \cdot f_m^{-1}(m_1^{-1}(x)) \quad , \quad x \in \triangle_1 \\ A_2 \cdot f_m^{-1}(m_2^{-1}(x)) + A_1 \quad , \quad x \in \triangle_2 \\ A_3 \cdot f_m^{-1}(m_3^{-1}(x)) + A_1 + A_2 \quad , \quad x \in \triangle_3 \\ A_4 \cdot f_m^{-1}(m_4^{-1}(x)) + A_1 + A_2 + A_3 \quad , \quad x \in \triangle_4 \end{cases} \tag{5.17}$$

$$= A_j \cdot f_m^{-1}(m_j^{-1}(x)) + \sum_{l<j} A_l \quad , \quad x \in \triangle_j. \tag{5.18}$$

The computation of the inverse of Moore's curve is considered in 5.1.2. It remains to define the mappings:

$$m_j^{-1} : \triangle_j \to [0,1]^2 \quad , \quad j \in \{1,2,3,4\}. \tag{5.19}$$

Since $(m_j)_{j=1...4}$ depend on M, it is rather difficult to find analytical mappings (and on the fly, since M is not fixed). Therefore, we decided to divide this step into two computation steps. First we map the triangles $(\triangle_j)_{j=1...4}$ to a reference triangle $\triangle_S$ defined by the vertices $(0,0),(0.5,1),(1,0)$. We denote this mapping with $n_j : \triangle_j \to \triangle_S \quad , \forall j \in \{1,2,3,4\}$. Then we apply a function $\varphi : \triangle_S \to [0,1]^2$. We precomputed φ on a grid and stored it in a file so that function values can be interpolated. n_j is of the form:

$$n_j(x) = A^j \cdot x + b^j \tag{5.20}$$

$$= \begin{pmatrix} a_{11}^j & a_{12}^j \\ a_{21}^j & a_{22}^j \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1^j \\ b_2^j \end{pmatrix} \tag{5.21}$$

$$= \begin{pmatrix} a_{11}^j & a_{12}^j & b_1^j \\ a_{21}^j & a_{22}^j & b_2^j \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}, \tag{5.22}$$

Matrix A^j rotates and stretches the triangle $\triangle_j$, and the vector b^j translates the triangle. If P_j, Q_j, M (labeled counterclockwise) denote the corner points of triangle $\triangle_j$, we obtain the conditions:

$$n_j(P_j) = (0,0) \tag{5.23}$$

$$n_j(Q_j) = (1,0) \tag{5.24}$$

$$n_j(M) = (0.5,1), \tag{5.25}$$

and we compute A^j and b^j as a solution of two small linear equation systems (3×3 in 2D, 4×4 in 3D).

For mapping $\triangle_S$ to $[0,1]^2$ we used the mapping φ as displayed in figure 5.4:

We computed φ as the solution of a Laplacian equation:

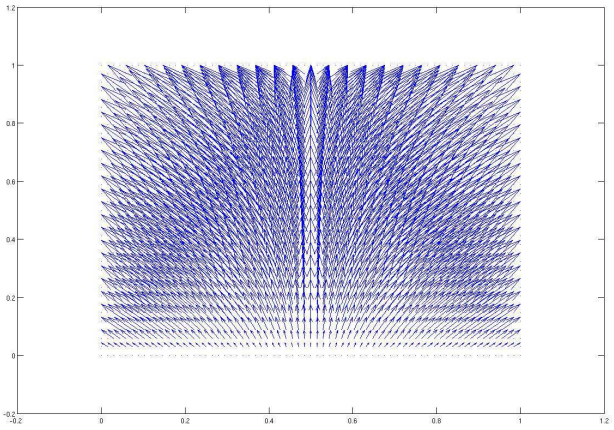


Figure 5.4: Displacement of smooth mapping from $\triangle_S$ to the unit square.

$$\begin{aligned} \Delta\varphi &= 0, \\ \partial\varphi(x) &= \begin{cases} id \quad , & x_2 = 0 \\ (0, \frac{3}{2}x_1), & 0 < x_2 < \frac{2}{3} \text{ and } x_1 < 0.5 \\ (1, 3 - 3x_1), & 0 < x_2 < \frac{2}{3} \text{ and } x_1 \geq 0.5 \\ (3x_1 - 1, 1), & x_2 \geq \frac{2}{3} \end{cases} . \end{aligned} \tag{5.26}$$

The boundary conditions are set up such that the boundary of the triangle is mapped continuously to the boundary of the unit square.

System 5.26 is discretized by finite differences. The unknowns of the triangle are collected bottom-up left to right. The grid spacings h_x, h_y are chosen such that $h_y = 2 \cdot h_x$, since this way the boundary points of all discrete y-coordinates $0 \leq k \cdot h_y \leq 1$ are aligned with the grid.

The resulting linear equation system has the non-zero structure as displayed in figure 5.5. It is diagonally dominant but not symmetric.

We solve this system with an algebraic multigrid solver implemented in *hypre* [1]. The discrete solution is stored in a file and via an interpolation φ can be computed for any $x \in \triangle_S$.

[1] http://www.llnl.gov/CASC/linear_solvers/

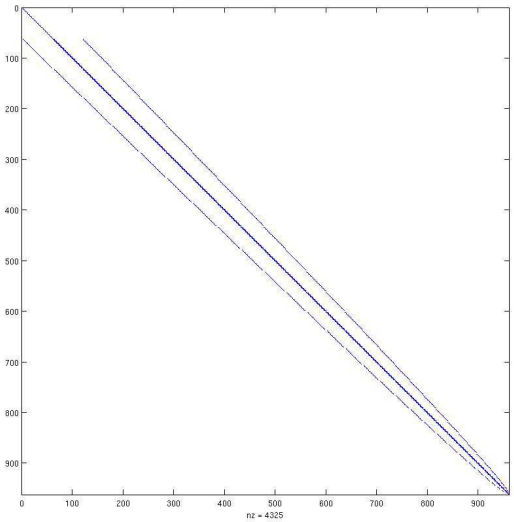


Figure 5.5: Non-zero structure of the arising linear equation system.

6 Test cases

Before discussing test cases that demonstrate the applicability of our two-constraint decomposition method, in section 6.1 we provide a test case that verifies the edge-cut growth rate in dependence of σ and the dimensionality as it is discussed in section 4.4.

The subsequent test cases compare our two-constraint method to MeTiS with respect to quality and speed. All test cases are in 3D. In the appendix A.1 we provide a table that itemizes the meshes that are applied in this section. In all the test cases we assume $\omega_M = \frac{1}{N}$, which corresponds to equal workload for each tetrahedron for the solution of the Maxwell equations. This is for the fact that primarily we intend particle simulations with equal polynomial degrees for all tetrahedra. We executed test cases with slightly varying ω_M and could not observe that the difficulties of the problems change significantly for MeTiS or for our two-constraint algorithm.

We base the test cases on different triangulations. We can not identify any geometries that are in any sense more difficult or easier to handle for our two-constraint algorithm. The difficulty rather stems from the combination of geometry and particle distribution in the geometry. We nevertheless apply different geometries to exclude that certain geometries influence the quality of the decomposition algorithms significantly.

Here it is important to mention that we do not consider the time for computing the inverse of SFC mappings and the time for reordering the tetrahedra with respect to their indices. In the setting of a full particle simulation we assume that the input meshes (that are stored in a file) are already reordered with respect to some SFC.

The test environment uses MeTiS as a library. The two-constraint decomposition method of MeTiS is called, which operates on the dual graphs of our meshes. The time for constructing dual graphs is neither considered, since they can also be precomputed.

The result of a DD by MeTiS is an array of size N (number of graph vertices) that states the domain decomposition as defined in 2.1. In order to compare the two methods, we generated the same interface to our two-constraint algorithm as the interface to the MeTiS library. Furthermore, we return the results of our two-constraint method in the same way MeTiS does. Note that for implementations in particle simulations this is not optimal. Rather, an indication of the DD in terms of intervals (e.g. partition P_i consists of tetrahedra $\{\tau_{j_1} \dots \tau_{k_1}\}$, $\{\tau_{j_2} \dots \tau_{k_2}\} \dots$) is more beneficial in terms of data organization, since the memory requirements to store the DD are extremely reduced [1]. In the following we discuss the quality of the resulting decompositions as well as their runtimes. We do not discuss implementation details or technical issues of the methods.

[1] If we store a DD as the result of our two-constraint algorithm as a list of separating intervals, this requires $O(\sigma \cdot N_P)$, since we only have to store partition boundaries (if stored as an array as returned by MeTiS, this is $O(N)$).

6.1 Verification of edge-cut growth rate in dependence of σ and the dimensionality

In section 4.4 we provide an estimation for the growth rate of the edge-cut in dependency of σ and the dimensionality of the problem. Equation 4.37 states that the edge-cut of a decomposition into N_P parts grows such that:

$$\text{“edge-cut of decomposition”} \in O(\sigma^{\frac{1}{d}}). \tag{6.1}$$

We verify this in the following. We base the experiments on the two meshes `square` and `cube`, compare appendix A.1. `square` is a 2D Delaunay triangulation of the unit square, `cube` is a natural extension of `square` to 3 dimensions, a triangulation of the unit cube.

6.1.1 A simple and homogeneous example

In a first experiment we intend to pose a simple decomposition problem such that all subparts of a partition cover about the same index range and the subparts are of about the same volume. We achieve this by setting $\omega_M = \omega_P = \frac{1}{N}$, which actually corresponds to a single-constraint problem, since balancing ω_P automatically balances ω_M. Here we chose $N_P = 4$ and base the algorithm on the Hilbert curve. With this experiment we intend to setup conditions that are close to the assumptions we made in the derivation of the growth rate in 4.4. The resulting subparts in this experiment are similarly shaped, since we selected $N_P = 4$ (Hilbert curve's inherent quartering), and the underlying geometries are such that they fully populate the index spaces in a homogeneous way. Furthermore, the edge-cut should be represented by the surface sizes quite well, since we created the meshes `square` and `cube` by triangulation tools that fulfill the demands of similarly sized and shaped triangles/tetrahedra.

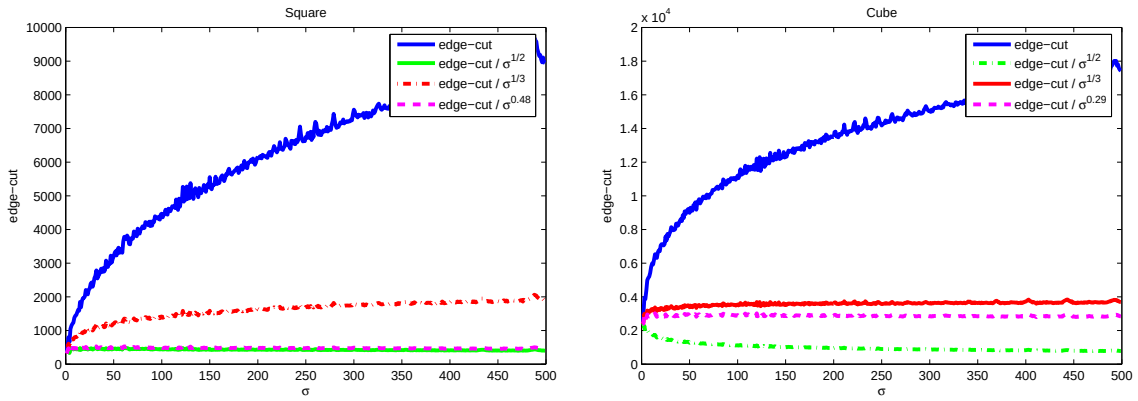


Figure 6.1: Edge-cuts and normalized edge-cuts in dependency of σ for a homogeneous decomposition problem of the meshes `square` (left) and `cube` (right).

Figure 6.1 shows the dependencies of the edge-cut in function of σ for the meshes `square` and `cube`. The blue lines represent the absolute edge-cut as we defined it in definition 2.4. The green lines display the edge-cut divided by $\sqrt{\sigma}$, and the red lines display the edge-cut

divided by $\sqrt[3]{\sigma}$. If the edge-cut of an experiment grows with $\sigma^{\frac{1}{d}}$ then the corresponding normalized edge-cut should resemble a straight line. The magenta colored line displays the edge-cut normalized by σ^l, l is chosen such that the edge-cut is proportional to σ^l as much as possible. We discuss the determination of l in 6.1.3.

The examples verify the predicted behavior. In both cases the actual growth rate (the magenta colored line) is better than the predicted growth rate, while in the two dimensional case the predicted growth rate and the actual growth rate nearly match.

We provide a second example that is intended to show that the growth rates also hold for heterogeneous and more difficult problems.

6.1.2 A difficult and heterogeneous example

Here we partition the meshes `square` and `cube` into $N_P = 21$ parts. We chose a highly heterogeneous particle distribution such that the balancing is challenging for our algorithm. The algorithm produces subparts that do neither resemble each other in shape, nor are they of similar sizes.

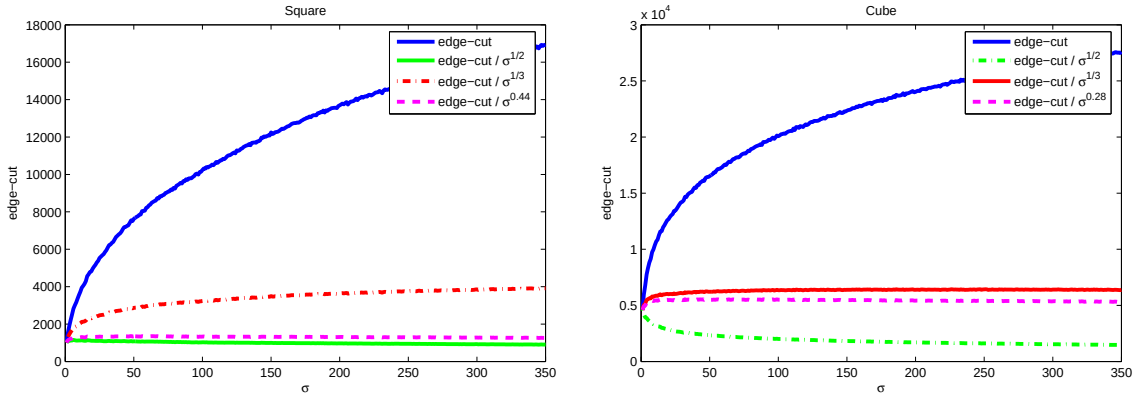


Figure 6.2: Edge-cut and normalized edge-cuts in dependency of σ for a heterogeneous decomposition problem of the meshes `square` (left) and `cube` (right).

Figure 6.2 shows that the predicted growth rates and the actual growth rates do not match as well as in the homogeneous testcase. For both geometries the growth rates are better than it is predicted by $O(\sigma^{\frac{1}{d}})$.

The example shows, however, that the orders of magnitude are met quite well, in particular the fact that the edge-cut in 3D grows more slowly than in 2D is demonstrated.

6.1.3 Computing the best fitting proportionality

We briefly explain how we compute the best fitting proportionality. Given are the edge-cuts $\mu_e^\sigma(T)$ for $\sigma = 1 \dots N_\sigma$ and a mesh T. We search for an l such that [2]:

[2]Clearly we make a mistake in assuming 6.3, however, for a rough estimation assumption 6.3 is sufficient.

$$\mu_e^\sigma(T) \in \theta(\sigma^l) \Rightarrow \tag{6.2}$$
$$\mu_e^\sigma(T) \approx C \cdot \sigma^l, \tag{6.3}$$

is satisfied best, hence we minimize the error

$$\sum_{\sigma=1}^{N_\sigma} |\frac{\mu_e^\sigma(T)}{\sigma^l} - C| \rightarrow \min \tag{6.4}$$

or equivalently

$$\sum_{\sigma=1}^{N_\sigma} \left(\frac{\mu_e^\sigma(T)}{\sigma^l} - C\right)^2 \rightarrow \min. \tag{6.5}$$

By differentiating 6.5 with respect to C, we get the condition

$$C = \frac{\sum_{\sigma=1}^{N_\sigma} \mu_e^\sigma(T)}{\sigma^l \cdot N_\sigma}. \tag{6.6}$$

A simple Matlab script numerically searches the best fitting l that minimizes 6.5.

6.2 Full particle coverage of a cylinder

The following example is based on the mesh `cea-cylinder`, compare appendix A.1. We consider a particle distribution as displayed in figure 6.3.

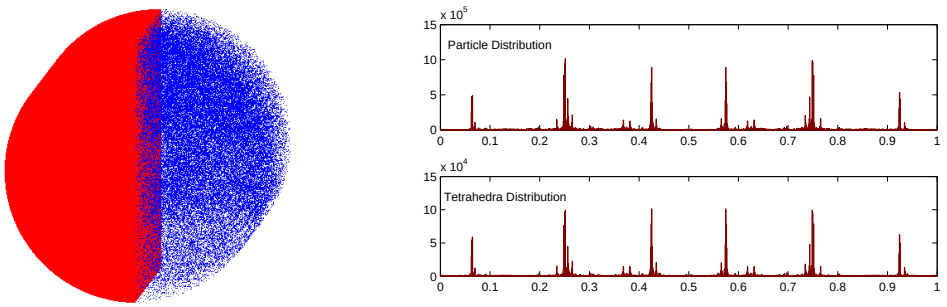


Figure 6.3: Decomposition problem based on mesh `cea-cylinder`, for the visualization here the right half is cut off. Each tetrahedron contains at least one particle. The distributions are provided in index space of Moore's curve.

The geometry, as it is the case of all geometries we consider, is a subset of $[0,1]^3$. We determine the numbers of particles per tetrahedron τ_i by:

$$(x_i, y_i, z_i) = cg(\tau_i) \tag{6.7}$$
$$\#\text{ part. in } \tau_i = \lfloor -rand() \cdot \sqrt{(x_i - 0.5)^2 + (y_i - 0.5)^2} \cdot 4.75 + 5 + 10 y_i \rfloor, \tag{6.8}$$

rand returns a random number in $[0, 1]$. The distribution is independent of the z-coordinate and tetrahedra with a large y coordinate contain more particles than tetrahedra with a small y coordinate, this property can be seen in figure 6.3. The distribution is selected such that each tetrahedron contains between 1 and 15 particles.

6.2.1 Comparison: MeTiS - two-constraint SFC

Here we provide a comparison of our two-constraint algorithm with MeTiS in terms of quality (balancing of weights, edge-cut) and speed. We perform decompositions into 2 to 512 parts. The provided results are obtained with our algorithm based on Moore's curve. Basing our algorithm on Hilbert's curve leads to similar results.

Since MeTiS can be used with different balancing constraints (compare to sections 3.1 and 3.2) and the two-constraint SFC algorithm can be used with different σ values, it is not evident how to set up these parameters to provide a meaningful comparison. We decided to allow an imbalance of 1.03 for all test cases, that is for a decomposition d it holds $\mu_w(\omega_P, d) \leq 1.03$ and $\mu_w(\omega_M, d) \leq 1.03$. To set up the appropriate σ to achieve these balancing constraints, we start with $\sigma = 1$ and increment σ until we reach the desired balancing. For the measuring of execution time of our two-constraint algorithm we just consider the runtime with the correctly chosen σ. The results are shown in figure 6.4.

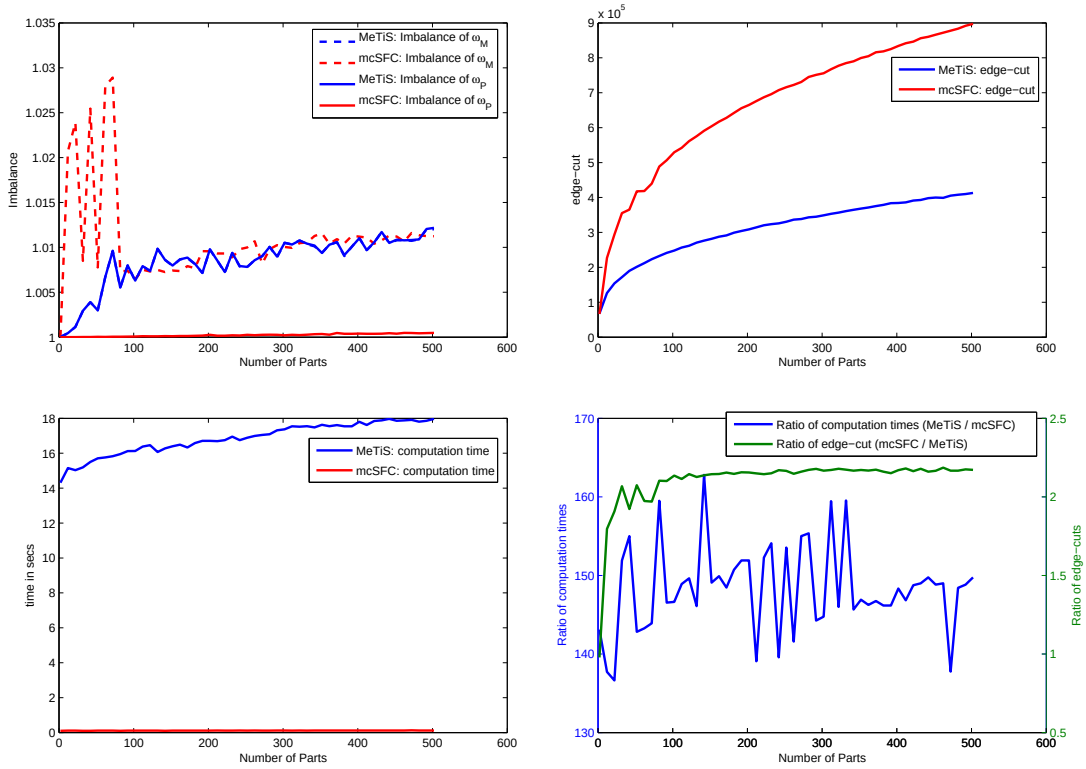


Figure 6.4: Results for decomposition problem based on mesh `cea-cylinder`.

In order to achieve the demanded balancing, σ values of 5 or 6 are required for all numbers of partitions ≥ 82. For smaller numbers of partitions smaller σ values are already sufficient.

The upper left figure shows the imbalances of the weights ω_M and ω_P. Even though we allowed MeTiS to produce partitions with an imbalance of 1.03, MeTiS produced partitions with significantly better balancings. The imbalance of ω_M produced by MeTiS is nearly equal to the imbalance of ω_P, the plot of ω_M is hidden behind the imbalance of ω_P.

The figures show the perfect balancing of ω_P achieved by our algorithm. For small numbers of partitions smaller σ values were chosen, and hence a higher imbalance for ω_M occurred to the benefit of smaller edge-cuts as displayed in the upper right figure. For 512 parts the edge-cut produced by our algorithm is about 2.15 times worse than the edge-cut produced by MeTiS as can be seen in the bottom right figure. This is remarkable since σ values of $5, 6$ are required, which means that partitions consist of 5 or 6 disconnected subpartitions. This is due to the fact that the edge-cut grows in $O(\sigma^{\frac{1}{3}})$ as discussed in section 4.4 and the test case of section 6.1.

In terms of computation time the two-constraint algorithm took between 0.1 and 0.12 seconds while MeTiS took between 14 and 18 seconds. Here the two-constraint algorithm is between 140 and 160 times faster than MeTiS.

6.2.2 Evolution of imbalance with respect to σ and N_P

We further measured the imbalance of ω_M with respect to σ and the number of parts N_P. The result is displayed in figure 6.5.

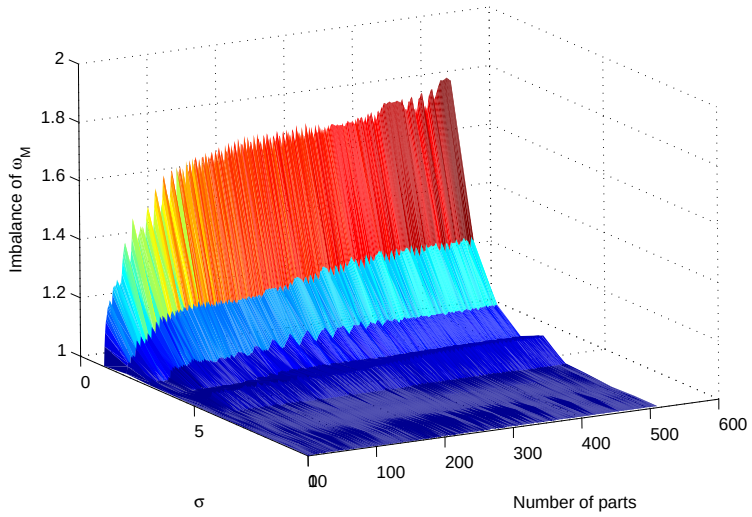


Figure 6.5: Result: Imbalance with respect to σ

Remarkable here is the quick diminishment of imbalance by only slight increases of σ. Since in general small σ values are sufficient for good balancings, this DD problem can be considered an easy problem.

6.3 Small particle cloud in cube

Here we consider the mesh `cea-cube`, compare appendix A.1. We concentrated 91444 particles in a particle cloud as displayed in figure 6.7.

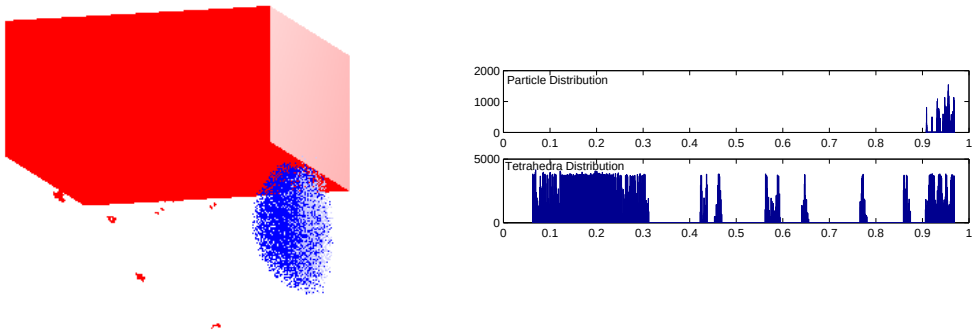


Figure 6.6: Decomposition problem based on mesh `cea-cube`. For the visualization here the bottom half is cut off. The distributions are provided in index space of Hilbert's curve.

This setting poses a rather difficult problem. The example contains 1220192 tetrahedra, but only 36473 tetrahedra contain particles. Hence, if we decompose the example into 100 parts, each part obtains approximately 364 tetrahedra of those that contain particles (and hence approximately 914 particles). These are very low numbers, and in general we expect particle simulations with many more particles. Nevertheless, we picked the example because it is somewhat an exemplar that gave rise to the development of the mixed-constraint version of our two-constraint algorithm.

6.3.1 Comparison: MeTiS - mixed-constraint SFC

Here the mixed-constraint algorithm suits much better than the two-constraint algorithm, since only few tetrahedra contain particles. We discuss decompositions into 2 to 256 parts, at most 256 parts, since the problem sizes are already very small.

The mixed-constraint algorithm first applies the two-constraint algorithm in the interval $[0.91, 0.97]$ to balance ω_P. In a second step the imbalance induced by this decomposition is balanced by a single-constraint decomposition of the remaining domain, which takes this imbalance into account. For high numbers of processors the induced imbalance by the first decomposition is rather high, and it is not possible to balance it with the second decomposition. For this reason increased σ values for the first decomposition have to be applied. Results are displayed in figure 6.7.

Firstly, the high imbalance of ω_P of the mixed-constraint algorithm is remarkable here, since ω_P should be balanced perfectly and automatically. This is due to the fact that the number of tetrahedra that contain particles is small. These tetrahedra are decomposed into subparts that contain only a couple of tetrahedra (5-15) and hence a small weight ω_P. In these small orders of magnitudes small absolute imbalances are already noticeable relative imbalances. During the reunification phase subparts are combined, and this procedure even

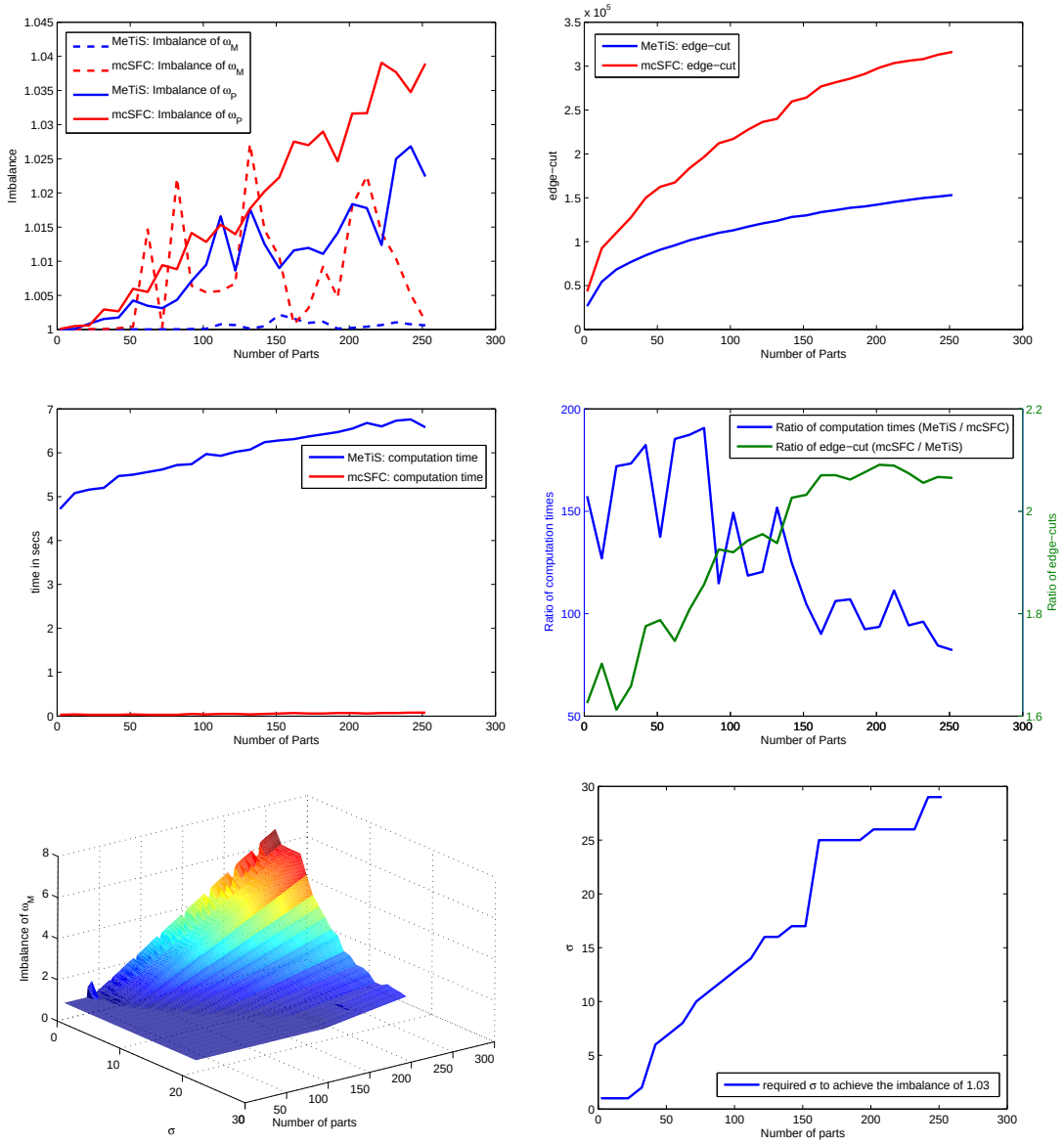


Figure 6.7: Results for decomposition problem based on mesh `cea-cube`

amplifies these imbalances (since the reunification does not consider ω_P). As a result we obtain imbalances of up to 1.04. This effect does not occur if the problem sizes, that is the number of tetrahedra and the number of particles, is sufficiently large, as it is the case for realistic PIC simulation settings.

Secondly, it is worth noticing that, even though high σ values are required to achieve the imbalance of 1.03, the edge-cuts are not noteworthily worse than in the test case of section 6.2, where smaller σ values were sufficient. The simple explanation is that a final partition consists of $\sigma - 1$ small subparts that are located in the particle area and one huge subpart outside the particle area. The edge-cuts of the small subparts are hence small and the overall edge-cut is mainly determined by the edge-cut of the area without particles. Note that in the prior example (in section 6.2) all subparts are of similar sizes. Hence, here large σ values do not contribute to the edge-cut as strongly as large σ values in the prior example.

The ratio of computation time of MeTiS and the mixed-constraint algorithm decreases, while increasing the number of parts. The figures show that a larger σ is required for large numbers of partitions. The runtime of RH2 grows linearly with σ, hence the total runtime of the mixed-constraint algorithm increases.

6.4 Benefits of problem-specific curves

We consider another particle cloud example based on mesh `cylinder` (compare appendix A.1) as displayed in figure 6.8.

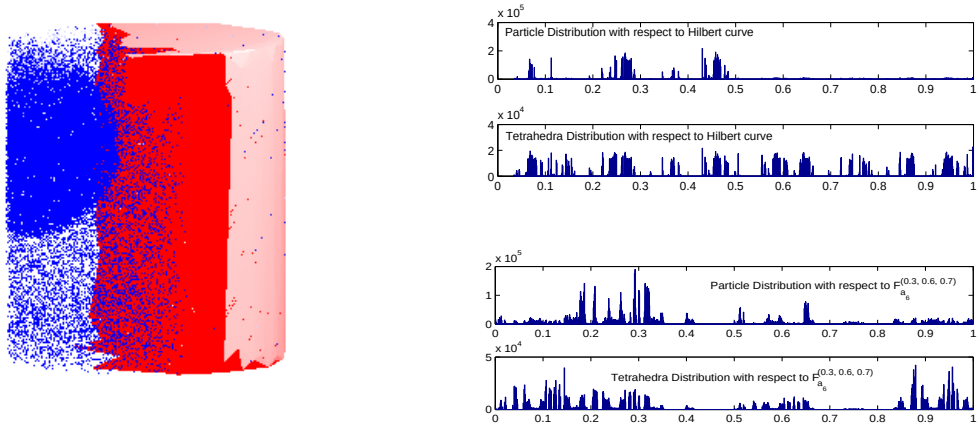


Figure 6.8: Particle cloud in cylinder: Distributions are provided in index space of Hilbert's curve and in index space of the problem-specific curve $F_{a6}^{(0.3,0.6,0.7)}$.

Each tetrahedron in the sphere with center $M = (0.3, 0.6, 0.7)$ and radius 0.2 contains 20 particles. The tetrahedra outside this sphere contain zero or one particle, each with a

probability of 50%. Figure 6.8 shows the distributions in index space of Hilbert's curve as well as in index space of the problem-specific curve $F_{a_6}^{(0.3,0.6,0.7)}$.

This is a typical setting where problem-specific curves lead to better results than the Hilbert curve does. We construct a problem specific curve around the center of the particle cloud, that is we consider $F_{a_6}^{(0.3,0.6,0.7)}$. As in the prior examples, we increase σ until we reach a weight imbalance smaller than 1.03. We compare MeTiS to the multi-constraint algorithm based on the Hilbert curve and the multi-constraint algorithm based on $F_{a_6}^{(0.3,0.6,0.7)}$. Results are shown in figure 6.9.

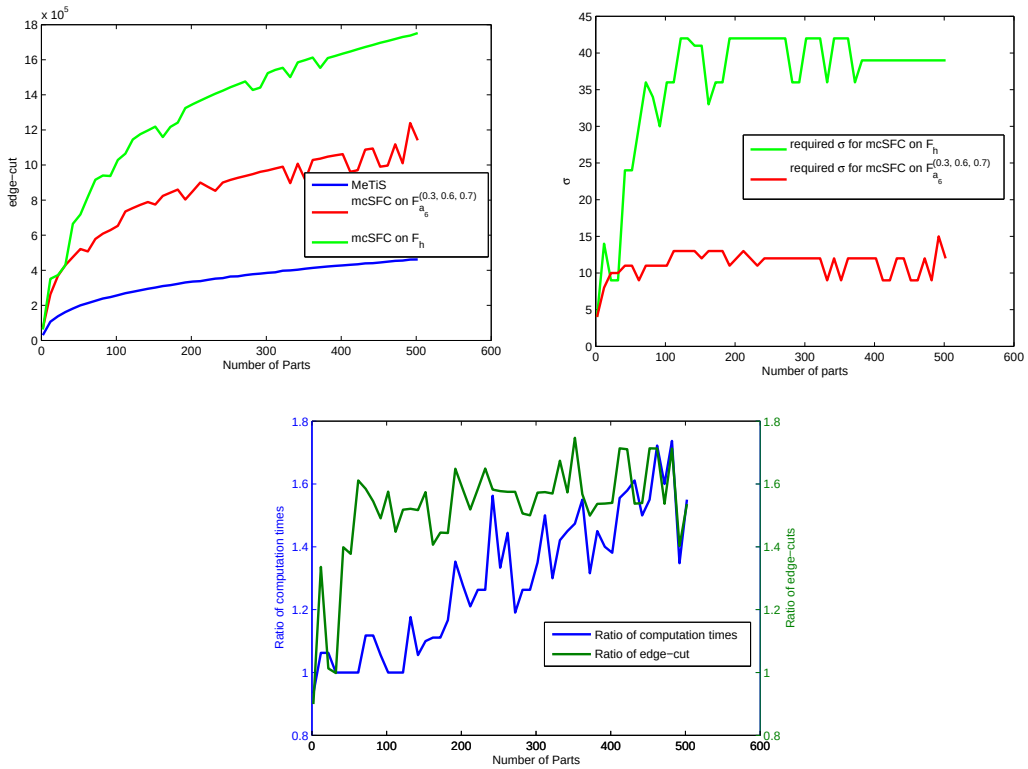


Figure 6.9: Results. The bottom figure compares the computation times and edge-cuts of the two-constraint algorithm based on Hilbert's curve with the two-constraint algorithm based on $F_{a_6}^{(0.3,0.6,0.7)}$.

The figures show that the two-constraint algorithm based on the Hilbert curve requires σ values up to 40, which leads to edge-cuts that are about 4 times as large as the edge-cuts provided by MeTiS. Basing the two-constraint algorithm on the problem-specific curve $F_{a_6}^{(0.3,0.6,0.7)}$ allows to keep σ small and we obtain similar results as in prior examples.

The bottom figure of figure 6.9 shows the ratio of edge-cuts and computation times of the two-constraint algorithm based on the Hilbert curve and based on $F_{a_6}^{(0.3,0.6,0.7)}$. Due to the large σ values the two-constraint algorithm based on $F_{a_6}^{(0.3,0.6,0.7)}$ is faster, since the runtime of RH2 grows with σ, and leads to smaller edge-cuts.

6.5 Particles in the mantle of cylinder - comparison of RH1 and RH2

We provide another example based on mesh `cea-cylinder` (compare A.1) which is also used in the test case of section 6.2. Particles are concentrated in the lateral mantle of a cylinder as displayed in figure 6.10. Tetrahedra in the mantle contain 10 particles, tetrahedra outside the mantle contain no particles with a probability of 90%, otherwise they contain one single particle.

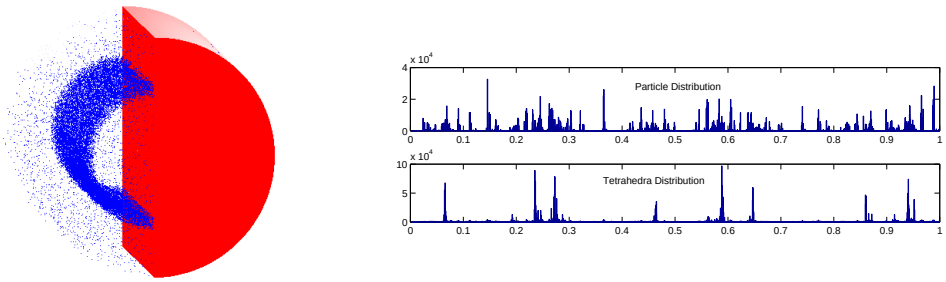


Figure 6.10: Mesh `cea-cylinder`. Particle are concentrated in the mantle of a cylinder.

Here we base our algorithm on the Hilbert curve. For demonstration purposes we also apply RH1 for REUNIFICATION and obtain the results as displayed in figure 6.11.

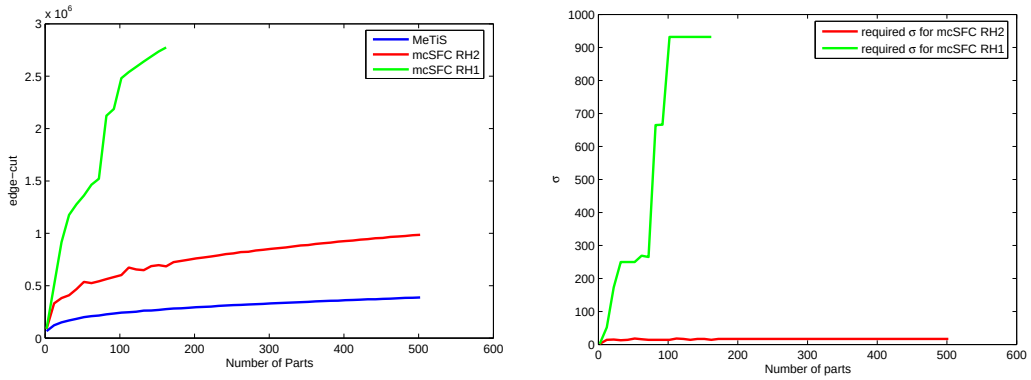


Figure 6.11: Results for decomposition problem: particles in mantle of a cylinder

If we base our two-constraint algorithm on RH2, we obtain similar results in comparison to MeTiS as in the other test cases concerning edge-cut, balancings, and execution time. The example rather demonstrates that the application of RH2 does not only improve the quality of the solutions but is absolutely necessary for our two-constraint method.

For $N_P \geq 102$ RH1 already requires that $\sigma = 932$, which leads to edge-cuts more than 10 times worse than the edge-cut produced by MeTiS. We interrupted the test series for RH1

at $N_P = 162$ simply because the evaluation and the search for the necessary σ requires an enormous amount of time.

7 Summary

The present report discusses a new DD method based on SFCs for two-constraint problems and its application in the context of particle simulations. The method requires the solution of a matrix splitting problem. We regarded this problem from two standpoints.

In section 4.2 we discussed a splitting with the minimal number of cuts possible such that the sum of imbalances is minimized. In other words, the number of cuts is fixed, and the objective is to minimize the imbalances. We showed that an algorithm that finds the best possible balancing of two weights is in $O(N^5 \cdot N_P^2)$. Since there are problems that can not be balanced by this algorithm, and due to the algorithm's bad complexity, we designed our new two-constraint algorithm as described in section 4.3.

Here we no longer minimize the number of cuts but rather split the matrix into multiple parts ($\sigma \cdot N_P$ parts) such that a balancing of both weights is possible. Note that our process of splitting and solving the REUNIFICATION problem does not lead to the optimal solution to the problem of finding the best solution given a maximal number of allowed cuts. In fact, it seems that there exist efficient algorithms to this problem, however, these algorithms have worse complexity than the problem of finding the best matrix splitting with the minimal number of cuts possible as discussed in 4.2. This observation makes our two-constraint algorithm appear in a different light: even though a best solution seems to be efficiently computable (but with bad complexity), we apply our two-constraint algorithm that computes worse results and even has to overcome the NP complete REUNIFICATION subproblem for the sake of fast computation times.

Furthermore, we showed that REUNIFICATION is NP complete and discussed the similarity of RH2 to LDM for the PARTITION problem. RH2 can be seen as an extension of LDM: RH2 substitutes two numbers a, b by some value $c \in [|a-b|, \ldots \max\{a,b\}]$, while LDM substitutes the two numbers by $|a-b|$.

Matrix W is the input to our two-constraint algorithm. The structure of W is strongly determined by the applied SFC. In section 4.5 we try to obtain matrix layouts that are easier to balance for our two-constraint algorithm by applying problem-specific SFCs. The test case in section 6.4 shows that there are applications for these curves.

Beside these theoretical considerations we provide test cases that compare our two-constraint DD method with MeTiS in chapter 6. In the following section 7.1 we discuss the two-constraint abilities of graph based multilevel schemes and compare them to our two-constraint SFC decomposition method. The report is closed by section 7.2 which discusses the sequel of the work.

7.1 Graph-based multilevel schemes versus Space Filling Curves for two-constraint particle simulations

7.1.1 Weight balancing

In general MeTiS and our two-constraint algorithm are able to balance both weights. The two-constraint algorithm exhibits the feature that it is able to balance one weight perfectly which is chosen to be the balancing of the particles throughout this work.

7.1.2 Edge-cut

In difficult settings it is necessary to apply large σ values in our two-constraint algorithm in order to achieve good balancing. The provided test cases show that for difficult settings the edge-cut of the two-constraint method may be up to four times worse [1] than the edge-cut induced by MeTiS, in most cases we observed edge-cuts that are between two and three times worse. Schamberger and Wierum reported in [31] that they observed edge-cuts for DD via SFCs up to seven times worse compared to MeTiS for single-constraint problems, and we do not exclude that this might happen as well for certain two-constraint problems.

As for single-constraint case MeTiS clearly outperforms SFC-based approaches.

7.1.3 Execution time and memory requirements

Since in our setting the computation of SFC indices can be precomputed as well as the sorting, our method performs extremely fast. Note that there is still room for improvements in the implementation. MeTiS is an optimized software package, while our two-constraint algorithm is in proof-of-concept state. Nevertheless, the timing measurements show the order of magnitude by which our two-constraint algorithm outperforms MeTiS. It is expectable that a fully engineered implementation leads to even better timing results. We can state that the two-constraint algorithm in our setting performs about 100 times faster than MeTiS.

Memory requirement measurements were not done, however, we expect similar results as for one-constraint problems, since our method does not make intensive additional use of memory consuming data structures.

7.1.4 Cache efficiency

In the setting of our two-constraint algorithm a partition consists of σ disconnected subpartitions. These subpartitions are stored cache efficiently in memory. For MeTiS it holds the same as in the single-constraint case, additional reordering strategies are necessary for cache efficient data storage. This is a clear advantage of our two-constraint algorithm.

7.2 Sequel

In the near future our two-constraint method will be further developed and incorporated into the development process of a full parallel PIC simulation with DDD for the Maxwell/Vlasov

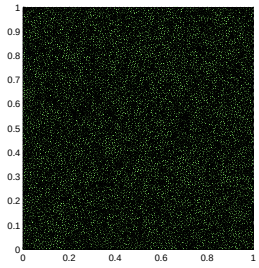
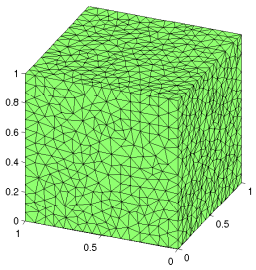
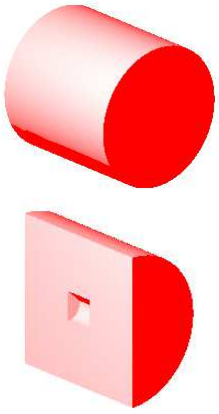
[1] E.g. test case 6.4 without the application of a problem-specific curve.

system. To obtain a performant particle simulation, the method of DD can not only be seen as an *external module* but rather influences the structure of the simulation software.

This implementation will allow a practical assessment of the applicability of the methods. We further plan to research redecomposition properties (average transfer volume induced by new decompositions, etc.) of the methods.

A Appendix

A.1 List of meshes

Name	Figure	# vert.	# ele.	Description
`square`		7900	15613	Delauney triangulation of the unit square, generated with the tool `triangle` [1], used in the test case in section 6.1
`cube`		3439	17727	Delauney triangulation of the unit cube, generated with the tool `tetgen` [2], used in the test case in section 6.1
`cea-cylinder`		562426	3245588	Cylinder with a cylinder cut out in its interior, provided by HOUPIC project partner CEA [3], used in the test cases in section 6.2 and 6.5

[1] http://www.cs.cmu.edu/~quake/triangle.html
[2] http://tetgen.berlios.de/
[3] www.cea.fr

Name	Figure	# vert.	# ele.	Description
`cea-cube`		211316	1220192	Cube with a cube cut out in its interior, provided by CEA, used in the test case of section 6.3
`cylinder`		500000	3370141	Cylinder, triangulated with the tool `tetgen`, used in the test case of section 6.4

Table A.1: Meshes used in the test cases of chapter 6.

Contents

Bibliography

[1] B. Aspvall, M. M. Halldórsson, and F. Manne. Approximations for the general block distribution of a matrix. *Theoretical Computer Science*, 262(1–2):145–160, 2001.

[2] M. Bader. Raumfüllende Kurven - begleitendes Skriptum zum entsprechenden Kapitel der Vorlesung Algorithmen des wissenschaftlichen Rechnens, Summer Term 2004. http://www5.in.tum.de/lehre/vorlesungen/algowiss/rfk.pdf.

[3] C. K. Birdsall and A. B. Langdon. *Plasma Physics via Computer simulation*. 1985.

[4] S. Boettcher and S. Mertens. Analysis of the Karmarkar-Karp differencing algorithm, 2008. to appear in European Physics Journal B, http://arxiv.org/abs/0802.4040.

[5] A.R. Butz. Alternative algorithm for Hilbert's space-filling curve. *IEEE Trans. Comput.*, 20(4):424–426, 1971.

[6] E. A. Carmona and L. J. Chandler. On parallel PIC versatility and the structure of parallel PIC approaches. *Concurrency - Practice and Experience*, 9(12):1377–1405, 1997.

[7] J.-P. Cioni D. Issautier, F. Poupaud and L. Fezoui. A 2-D Vlasov-Maxwell solver on unstructured meshes. *Proceedings of the Third International Conference on Mathematical and Numerical Aspects of Wave Propagation*, pages 187–204, 1995.

[8] D. Issautier and F. Poupaud and J.-P. Cioni and L. Fezoui. A 2-D Vlasov-Maxwell solver on unstructured meshes. *Third International Conference on Numerical Aspects of Wave Propagation*, 1997.

[9] C. Gotsman and M. Lindenbaum. The metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5(5):794–797, 1996.

[10] M. Grigni and F. Manne. On the complexity of the generalized block distribution. In *Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 319–326, 1996.

[11] B. Hayes. The easiest hard problem. *American Scientist*, 90(2), 2002.

[12] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000.

[13] B. Hendrickson and A. Pothen. Combinatorial scientific computing: The enabling power of discrete algorithms in computational science. In Michel J. Daydé, José M. L. M. Palma, Alvaro L. G. A. Coutinho, Esther Pacitti, and Joao Correia Lopes, editors, *VECPAR*, volume 4395 of *Lecture Notes in Computer Science*, pages 260–280. Springer, 2006.

[14] R. W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles*. 1981.

[15] Jan Hungershöfer and Jens-Michael Wierum. On the quality of partitions based on space-filling curves. In *ICCS '02: Proceedings of the International Conference on Computational Science-Part III*, pages 36–45, London, UK, 2002. Springer-Verlag.

[16] K.-C. Tseng J.-S. Wu. Parallel dsmc method using dynamic domain decomposition. *International Journal for Numerical Methods in Engineering*, 63:37–76, 2005.

[17] G. B. Jacobs and J. S. Hesthaven. High-order nodal discontinuous Galerkin particle-in-cell method on unstructured grids. *J. Comput. Phys.*, 214(1):96–121, 2006.

[18] G. Jin and J. Mellor-Crummey. Sfcgen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Trans. Math. Softw.*, 31(1):120–148, 2005.

[19] N. Karmarker and R. M. Karp. The differencing method of set partitioning. Technical report, Berkeley, CA, USA, 1983.

[20] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[21] G. Karypis and V. Kumar. Metis - a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices - user manual. online, 1998. http://glaros.dtc.umn.edu/gkhome/metis/metis/publications.

[22] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical report, University of Minnesota, Department of Computer Science, 1998. http://glaros.dtc.umn.edu/gkhome/node/90.

[23] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, pages 291–308, 1970.

[24] J. Lawder. Calculation of mappings between one and n-dimensional values using the Hilbert space-filling curve, 2000.

[25] S. Lanteri M. Bernacki and S. Piperno. Time-domain parallel simulation of heterogeneous wave propagation on unstructured grids using explicit, nondiffusive, discontinuous Galerkin methods. *Journal of Computational Acoustics*, 14(1):57–81, 2006.

[26] S. Lanteri M. Bernacki, L. Fezoui and S. Piperno. Parallel discontinuous Galerkin unstructured mesh solvers for the calculation of three-dimensional wave propagation problems. *Applied Mathematical Modelling*, 30:744–763, 2006.

[27] H. Meyerhenke, B. Monien, and T. Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. In *To appear in Proc. 22nd International Parallel and Distributed Processing Symposium, (IPDPS'08)*. IEEE Computer Society, 2008. Best Algorithms Paper Award.

[28] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Dover Publications, Inc., 1998.

[29] G. Rozenberg and A. Salomaa. *Mathematical Theory of L Systems*. Academic Press, Inc., Orlando, FL, USA, 1980.

[30] H. Sagan. *Space Filling Curves*. Springer-Verlag Gmbh, 1994.

[31] S. Schamberger and J.-M. Wierum. Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In *PaCT*, pages 165–179, 2003.

[32] B. Thidé. *Electromagnetic Field Theory*. online Book, 2007. http://www.plasma.uu.se/CED/Book/.

[33] C. Walshaw, M. Cross, and K. McManus. Multiphase mesh partitioning. *Appl. Math. Modelling*, 25(2):123–140, 2000.

[34] J.-S. Wu and K.-C. Tseng. Parallel DSMC method using dynamic domain decomposition. *International Journal for numerical methods in Engineering*, pages 37–76, 2005.

[35] G. Zumbusch. On the quality of space-filling curve induced partitions. *"Zeitschrift für angewandte Mathematik und Mechanik"*, 81(1):25–28, 2001.

[36] G. Zumbusch. *Parallel Multilevel Methods*. Teubner, 2003.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399